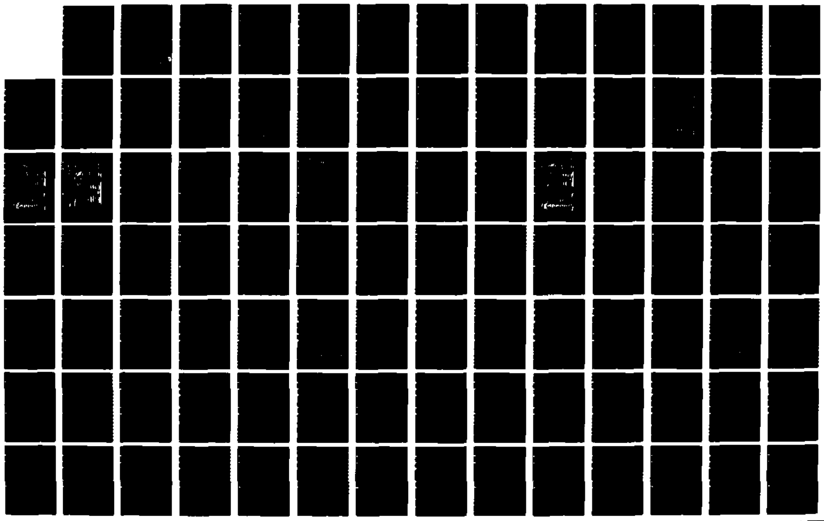
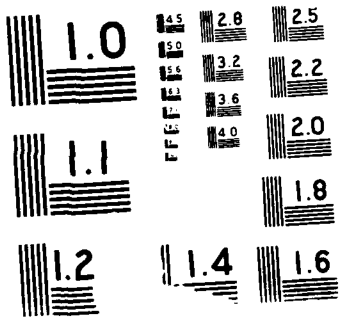
AD-A194 286   SIGNAL PROCESSING SOFTWARE FOR GROUND PENETRATING RADAR   1/3
USER'S MANAAL(U) SIGMA RESEARCH INC REDMOND WA
R LIEM ET AL MAR 88 NCEL-CR-88-010 N62474-85-C-4953

UNCLASSIFIED   F/G 17/9   NL




1.0
1.1
1.2
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

(4)

CR 88.010

March 1988

An Investigation Conducted By
Sigma Research, Inc.

Sponsored By Naval Facilities
Engineering Command

NCEL
Contract Report

SIGNAL PROCESSING SOFTWARE FOR GROUND PENETRATING RADAR USER'S MANUAL

ABSTRACT This is the user's manual for the signal processing software for reducing ground penetrating radar (GPR) data. The manual provides background information and instructions for operating the computer program. The developed program is based on the synthetic aperture focusing technique. Input data to the program consists of digitized sequential GPR scans from a linear survey. The format for the input data is specified in Appendix C. The output of the program are two-dimensional plots of the ground profile showing the stations and depth of the objects identified by the program. Features of the program include utilities to determine the velocity of propagation of the GPR signal and the location of the ground surface as well as semi-automatic and automatic processing of the data. The program is designed to operate on an IBM PC or compatible computer. Other hardware and supporting software requirements for operating the program are specified in Appendix B.

NAVAL CIVIL ENGINEERING LABORATORY PORT HUENEME CALIFORNIA 93043

METRIC CONVERSION FACTORS

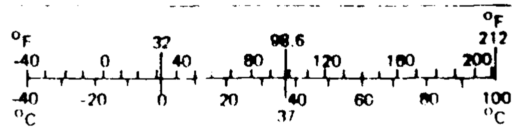
Approximate Conversions to Metric Measures

Symbol	When You Know	Multiply by	To Find	Symbol
		LENGTH		
in	inches	*2.5	centimeters	cm
ft	feet	30	centimeters	cm
yd	yards	0.9	meters	m
mi	miles	1.6	kilometers	km
		AREA		
in^2	square inches	6.5	square centimeters	cm^2
ft^2	square feet	0.09	square meters	m^2
yd^2	square yards	0.8	square meters	m^2
mi^2	square miles	2.6	square kilometers	km^2
	acres	0.4	hectares	ha
		MASS (weight)		
oz	ounces	28	grams	g
lb	pounds	0.45	kilograms	kg
	short tons (2,000 lb)	0.9	tonnes	t
		VOLUME		
tsp	teaspoons	5	milliliters	ml
Tbsp	tablespoons	15	milliliters	ml
fl oz	fluid ounces	30	milliliters	ml
c	cups	0.24	liters	l
pt	pints	0.47	liters	l
qt	quarts	0.95	liters	l
gal	gallons	3.8	liters	l
ft^3	cubic feet	0.03	cubic meters	m^3
yd^3	cubic yards	0.76	cubic meters	m^3
		TEMPERATURE (exact)		
°F	Fahrenheit temperature	5/9 (after subtracting 32)	Celsius temperature	°C

*1 in = 2.54 (exactly). For other exact conversions and more detailed tables, see NBS Misc. Publ. 286, Units of Weights and Measures, Price $2.25, SD Catalog No. C13.10:286.

Approximate Conversions from Metric Measures

Symbol	When You Know	Multiply by	To Find	Symbol
		LENGTH		
mm	millimeters	0.04	inches	in
cm	centimeters	0.4	inches	in
m	meters	3.3	feet	ft
m	meters	1.1	yards	yd
km	kilometers	0.6	miles	mi
		AREA		
cm^2	square centimeters	0.16	square inches	in^2
m^2	square meters	1.2	square yards	yd^2
km^2	square kilometers	0.4	square miles	mi^2
ha	hectares (10,000 m^2)	2.5	acres	
		MASS (weight)		
g	grams	0.035	ounces	oz
kg	kilograms	2.2	pounds	lb
t	tonnes (1,000 kg)	1.1	short tons	
		VOLUME		
ml	milliliters	0.03	fluid ounces	fl oz
l	liters	2.1	pints	pt
l	liters	1.06	quarts	qt
l	liters	0.26	gallons	gal
m^3	cubic meters	35	cubic feet	ft^3
m^3	cubic meters	1.3	cubic yards	yd^3
		TEMPERATURE (exact)		
°C	Celsius temperature	9/5 (then add 32)	Fahrenheit temperature	°F



Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER CR 88.010	2. GOVT ACCESSION NO	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Signal Processing Software for Ground Penetrating Radar User's Manual		5. TYPE OF REPORT & PERIOD COVERED Final Aug 1985 - Sep 1987 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Ronnie Liem and Thomas J. Davis		8. CONTRACT OR GRANT NUMBER(s) N62474-85-C-4953
9. PERFORMING ORGANIZATION NAME AND ADDRESS Sigma Research, Inc. 8710 148th Ave N.E. Redmond, WA 98052		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Y0995-01-004-420
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Civil Engineering Laboratory Port Hueneme, CA 93043-5003		12. REPORT DATE March 1988 13. NUMBER OF PAGES 244
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Naval Facilities Engineering Command 200 Stovall Street Alexandria, VA 22332-2300		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution is unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) ground penetrating radar, GPR, data reduction, image reconstruction, synthetic aperture focusing technique, SAFT, underground surveys, utility lines, obstacles		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This is the user's manual for the signal processing software for reducing ground penetrating radar (GPR) data. The manual provides background information and instructions for operating the computer program. The developed program is based on the synthetic aperture focusing technique. Input data to the		

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

program consists of digitized sequential GPR scans from a linear survey. The format for the input data is specified in Appendix C. The output of the program are two-dimensional plots of the ground profile showing the stations and depth of the objects identified by the program. Features of the program include utilities to determine the velocity of propagation of the GPR signal and the location of the ground surface as well as semi-automatic and automatic processing of the data. The program is designed to operate on an IBM PC or compatible computer. Other hardware and supporting software requirements for operating the program are specified in Appendix B.

TABLE OF CONTENTS

LIST OF FIGURES

1.0 INTRODUCTION

The primary purpose of this manual is to set forth instructions for operating the radar imaging software supplied by Sigma. This software generates high resolution, two-dimensional images of buried utilities by processing digitized sequential radar scans from a linear survey. The software program provide a number of user-friendly utilities, including velocity determination and ground surface identification, resulting in improved accuracy of target depth assessment.

The program uses Synthetic Aperture Focusing Techniques (SAFT) to reconstruct images. This process is described at length in Appendix A, and is implemented by using phased array techniques on a set of data synthesized by moving a single antenna.

2.0 SOFTWARE OVERVIEW

The software package is made up of seven programs that generate the menus for program execution and parameter inputs, and five programs that perform the data processing (Figure 2-1 and 2-2). The execution of the programs is fully menu driven. Prompts for additional inputs and operator required actions are displayed on the screen.

2.1 Highlight of Software Functions and Features

The seven menu programs and five execution programs can be generally summarized into six separate functions as follows:

2.1.1 SELECT

SELECT is the program execution selection program. In addition to the built in directory function it calls and transfers control to the programs below.

2.1.2 PREVIEW

Preview plots a two-dimensional view of the collected survey data. It allows the operator to obtain an overview of the data and decides what portions to process. Its output files are used by TRANSFER for writing a T-SAFT data file. Preview also provides a means of entering parameter values and station identification that would define and tag the data.

2.1.3 TRANSFER

TRANSFER is used to select a portion of the sequential data for T-SAFT image reconstruction. It rewrites the serial magnitude byte format of the digitized field data to a DC restored bipolar integer format.

2.1.4 T-SAFT

T-SAFT is the heart of the processing system. It takes the output files from TRANSFER and processes it to form high resolution bipolar SAFT reconstructed images (see Appendix A for a discussion of T-SAFT). The reconstructed image shows a two-dimensional depth and distance scaled plot of the underground target.

2.1.5 VELOCITY

VELOCITY is used to determine the velocity and time offset of the radar data. It is an interactive program using a curve fitting method. The goal is to find the best match between the hyperbolic radar signal from a target to a computer generated hyperbola. When matched the template hyperbola parameters are used to specify the radar data.

2.1.6 REVIEW

REVIEW is a vehicle for accessing the data file parameter header and the saved screen files. The accessed parameters are displayed on the screen and can be modified as desired. The screen file are files of plotted images

Figure 2-1.

NCEL RADAR DATA PROCESSING SOFTWARE OVERVIEW

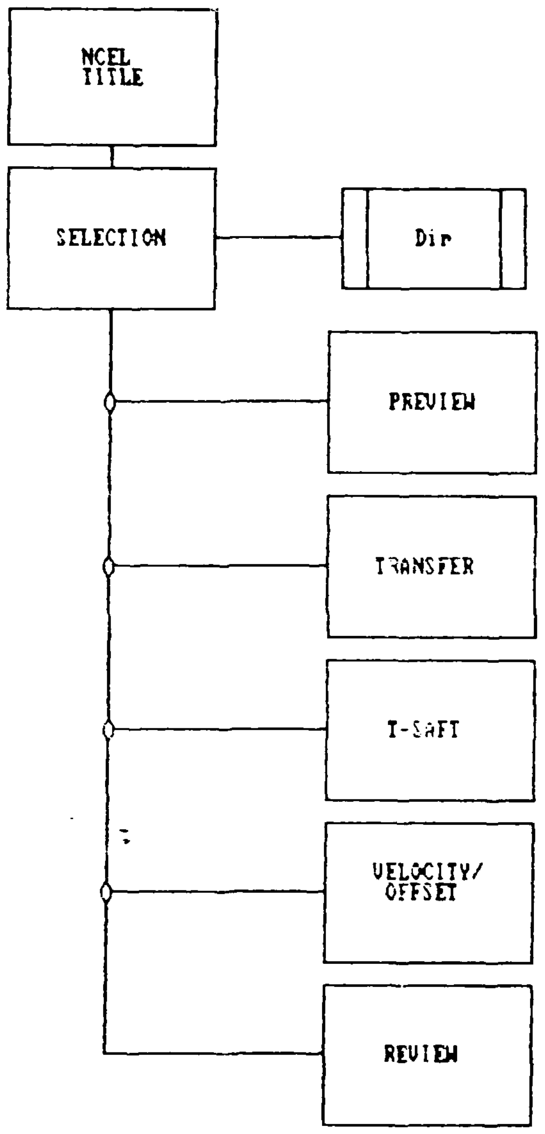
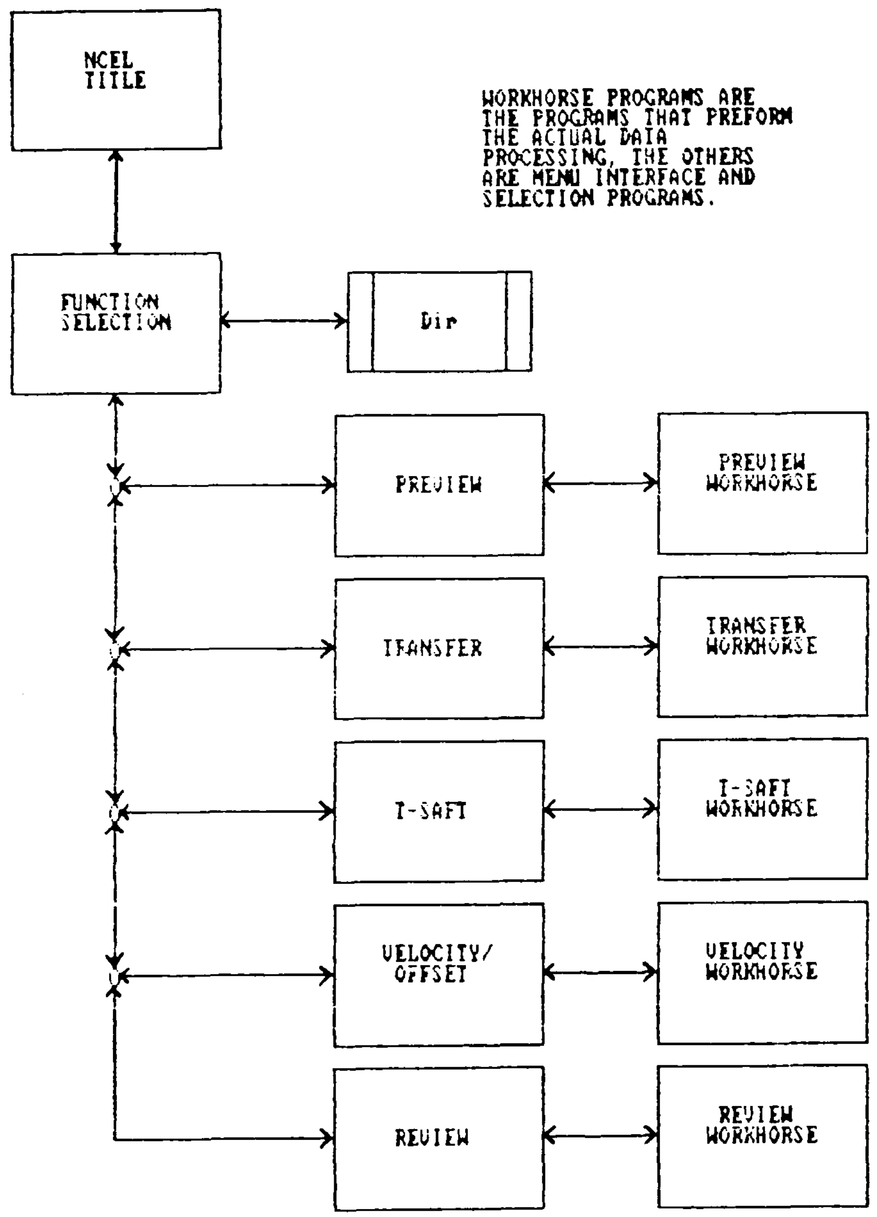


Figure 2-2.

SEMI-AUTOMATIC PROCESSING PROGRAM FLOW



generated by PREVIEW, TRANSFER, and T-SAFT. They are displayed on the screen and can be printed on the dot matrix printer.

2.1.7 AUTO PROCESS

Automatic data process program flow is shown in Figure 2-3. AUTO PROCESS features the automated processing of an entire diskfull of radar survey data. This function makes use of PREVIEW, TRANSFER, and T-SAFT to repeated work on consecutive segments of the radar survey to form a continuous non-overlapping reconstructed output image of the underground targets.

2.2 Data Files and Program Relationships

Figure 2-4 is introduced here to provide a frame of reference of the relationships between the input and output files and their processing algorithms. These relationships established early will help keep track of the data processing procedures.

This software package has been designed in a modular architecture. It provides a solid framework for menu driven GPR radar data processing. Enhancements to the system can be easily incorporated into the structure.

Figure 2-3.

FULLY-AUTOMATIC DATA PROCESSING PROGRAM FLOW

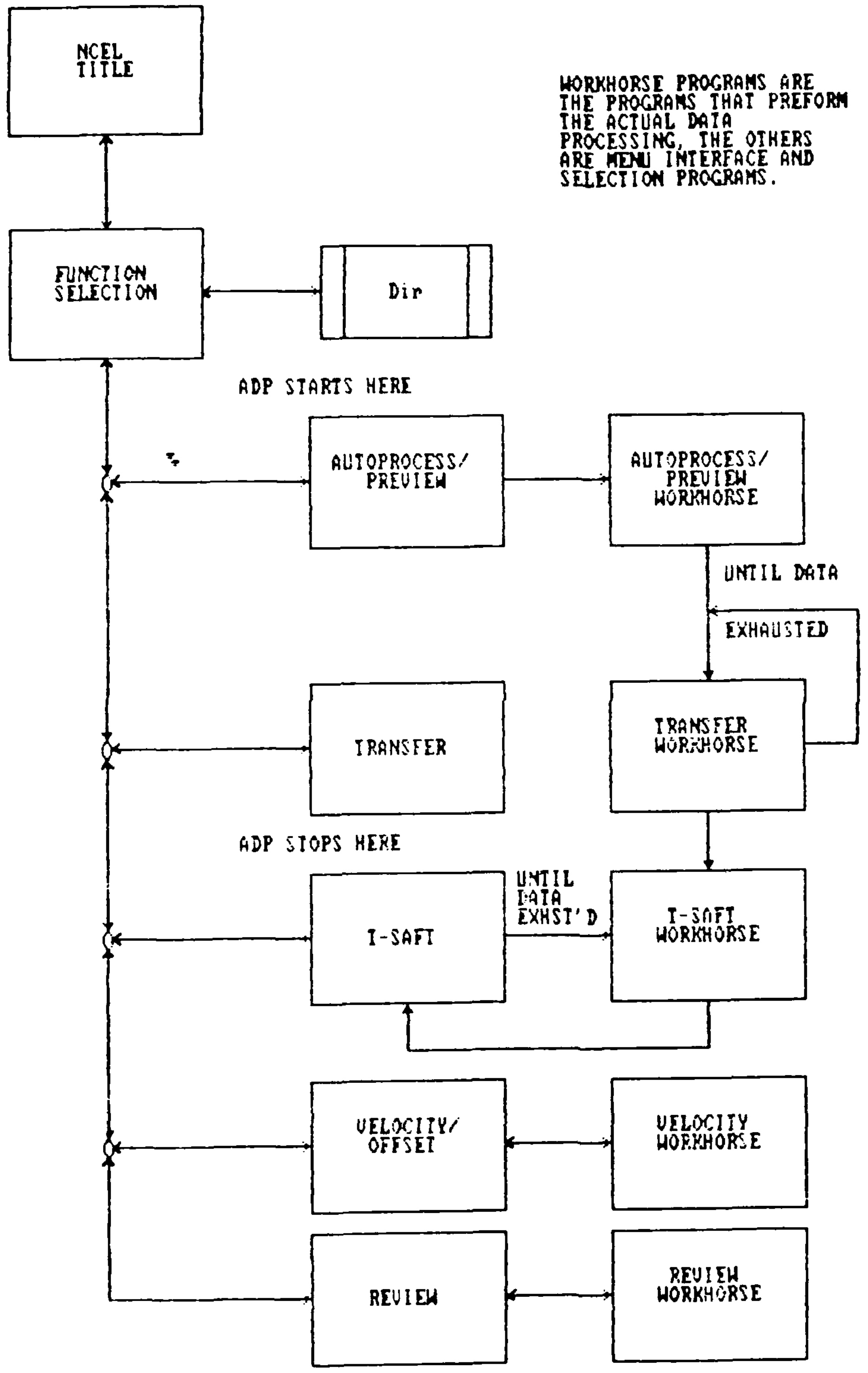
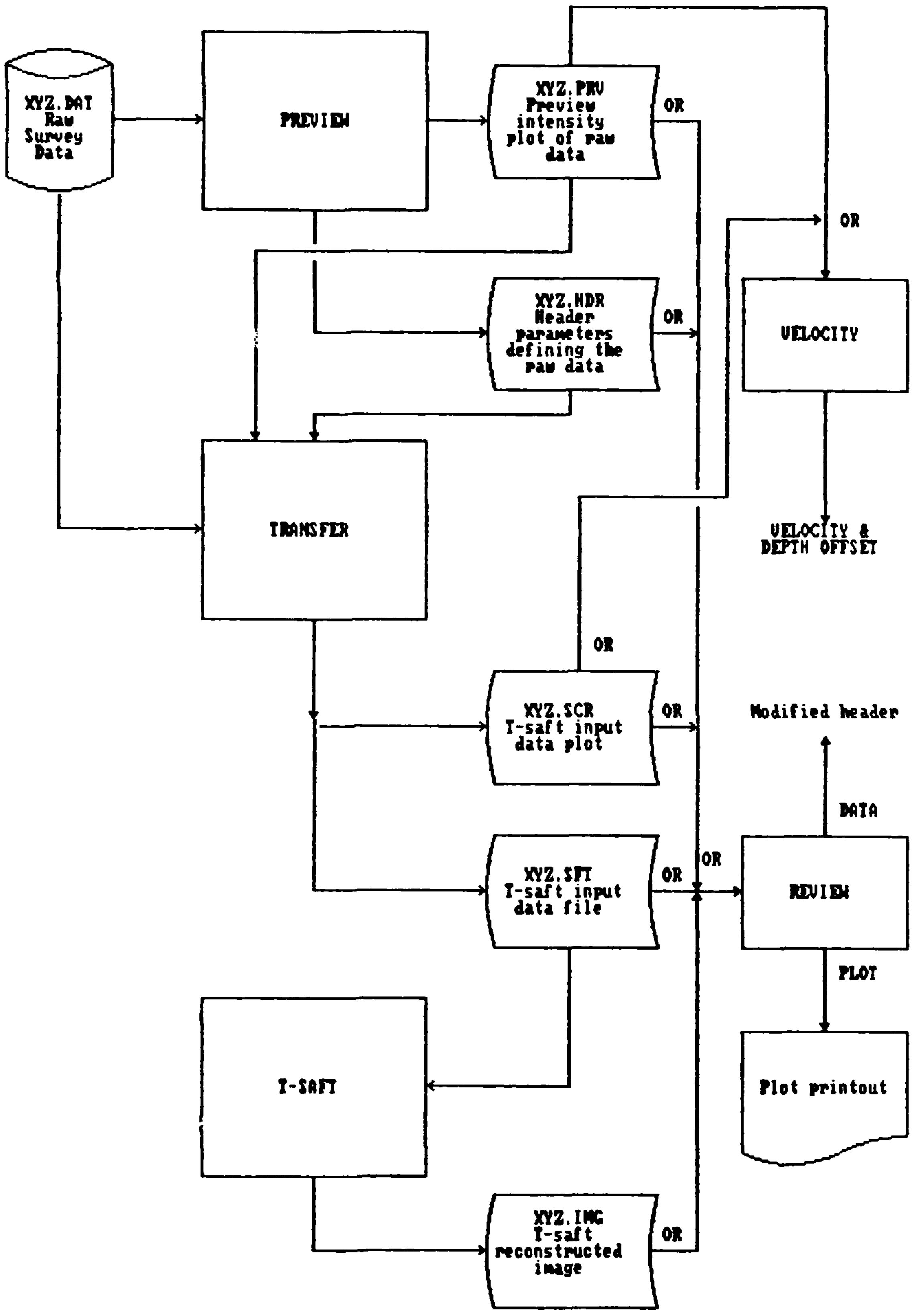


FIG 2-4 INPUT / OUTPUT FILES AND PROCESSING SOFTWARE RELATIONSHIP



XYZ.DAT Raw Survey Data
PREVIEW
XYZ.PRV Preview intensity plot of raw data
OR
OR
VELOCITY
XYZ.HDR Header parameters defining the raw data
OR
VELOCITY & DEPTH OFFSET
TRANSFER
OR
XYZ.SCR T-saft input data plot
OR
Modified header
DATA
XYZ.SFT T-saft input data file
OR
OR
REVIEW
PLOT
T-SAFT
Plot printout
XYZ.IMG T-saft reconstructed image
OR

3.0 OPERATIONS

The operation of the software program is menu driven. All required selection and parameter entries are prompted on the screen.

3.1 Software Installation

The radar data image processing should be installed in the default drive, preferably the hard disk drive for better access speed. Program installation is simply done by copying the execution files into default drive medium including the autoexe.bat file. The autoexe.bat file executes the graphics.com file of DOS when power to the machine is turned on. The graphics.com program is used by the radar imaging software to print a displayed graphic screen.

3.2 Common Menu Features

The PREVIEW menu is displayed in Figure 3-4 as a sample menu. In this menu and any other menus, the entries can be broken down into four categories: 1) Input file, 2) output file, 3) processing parameters, and 4) processing options. All these entries have default values that are displayed when the menu is first brought up. Any entry can be independently updated on the entire menu before the execution of the program. A blinking cursor is displayed at the first entry when the menu is first displayed. That entry can be changed by typing over it with the new value or character string. If no change is needed, simply depress <CR>, down arrow or right arrow for the next entry. To go back to the previous entry, press the up or left arrow. For numeric entry with a whole number and decimal portion, the entire number must be typed including a preceeding zero and a trailing zero. A special feature is implemented to check the existence of an input file. This is automatically carried out when the cursor is on the input file name and the <CR> key is depressed or when the 'F1' execution key is depressed regardless of cursor position. In T-SAFT when <CR> is depressed, another feature is invoked. The displayed parameters are updated by the parameters from the input files. Further modifications can be made by selecting the item of interest and entering a new value.

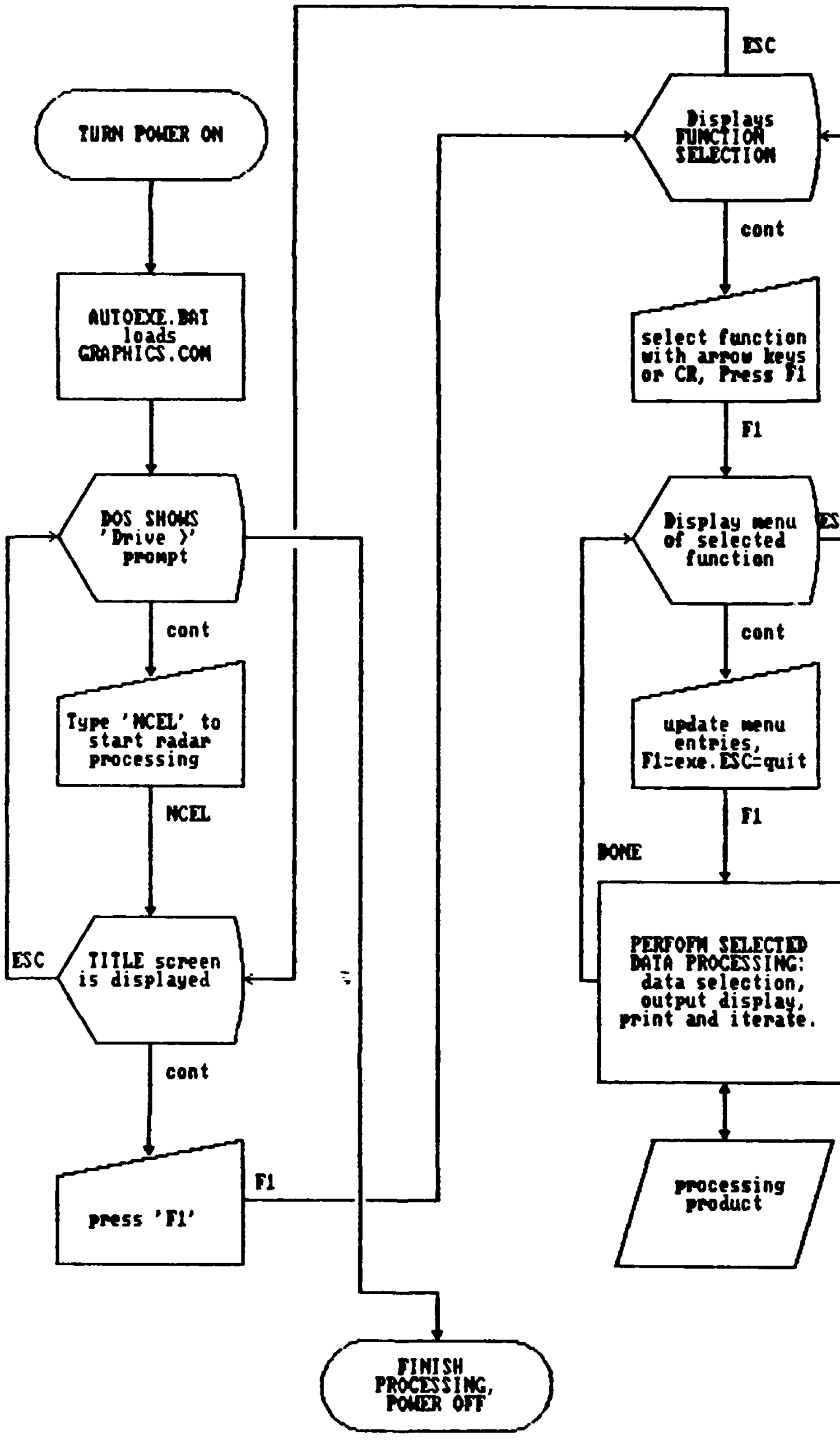
When the parameters and options have been reviewed the data processing program is executed by depressing 'F1'. Depending on the processing options involved, the software may need additional inputs while it is running. Prompts are displayed on the bottom of the screen when further inputs or operations are required.

In the processing option a choice is given to either print or not print the resultant output. In either case, the result is stored in the disk specified by the output data file.

<u>A word of caution: The operator must insure that there is sufficient disk space for output data files generated by the data processing software. Otherwise the disk write error will terminate the execution of the radar data processing software.</u>

Figure 3-1 shows an overview of the software execution. It captures the relationships between the operator actions and software responses.

FIG 3-1 NCEL RADAR PROCESSING SOFTWARE PROGRAM EXECUTION FLOW


TURN POWER ON
AUTOEXE.BAT loads GRAPHICS.COM
DOS SHOWS 'Drive >' prompt
cont
Type 'NCEL' to start radar processing
NCEL
TITLE screen is displayed
ESC
cont
press 'F1'
F1
FINISH PROCESSING, POWER OFF
ESC
Displays FUNCTION SELECTION
cont
select function with arrow keys or CR, Press F1
F1
Display menu of selected function
ESC
cont
update menu entries, F1=exe.ESC=quit
F1
DONE
PERFORM SELECTED DATA PROCESSING: data selection, output display, print and iterate.
processing product

3.3 Program Initiation

The NCEL radar data imaging software is started from the title screen by typing 'NCEL' followed by a carriage return <CR>. A title screen as shown in Figure 3-2 will appear. When this is displayed, two possible choices are available. Pressing 'F1' will bring up the function selection menu and 'ESC' will return control to DOS.

3.4 Function Selection

Function selection is invoked while the 'NCEL' title screen is displayed by pressing 'F1'. The function selection menu is shown in Figure 3-3.

The function selection menu displays seven processing choices. If none are desired, exiting from this menu back to the title screen is done by pressing 'ESC'. The function of each of these selections has been outlined in Section 2.1. A selection is made by moving the blinking cursor to a desired selection using the arrow keys or the <CR> key. When a selection has been made, that function is executed by pressing 'F1'. Another menu pertinent to the software function selected is displayed. These functions will be discussed in the order listed in the selection menu. With the exception of the directory, each function has a menu program and a data processing program. The menu program has names identical to the function and the data processing programs have similar names ending in 'WK'.

3.5 PREVIEW

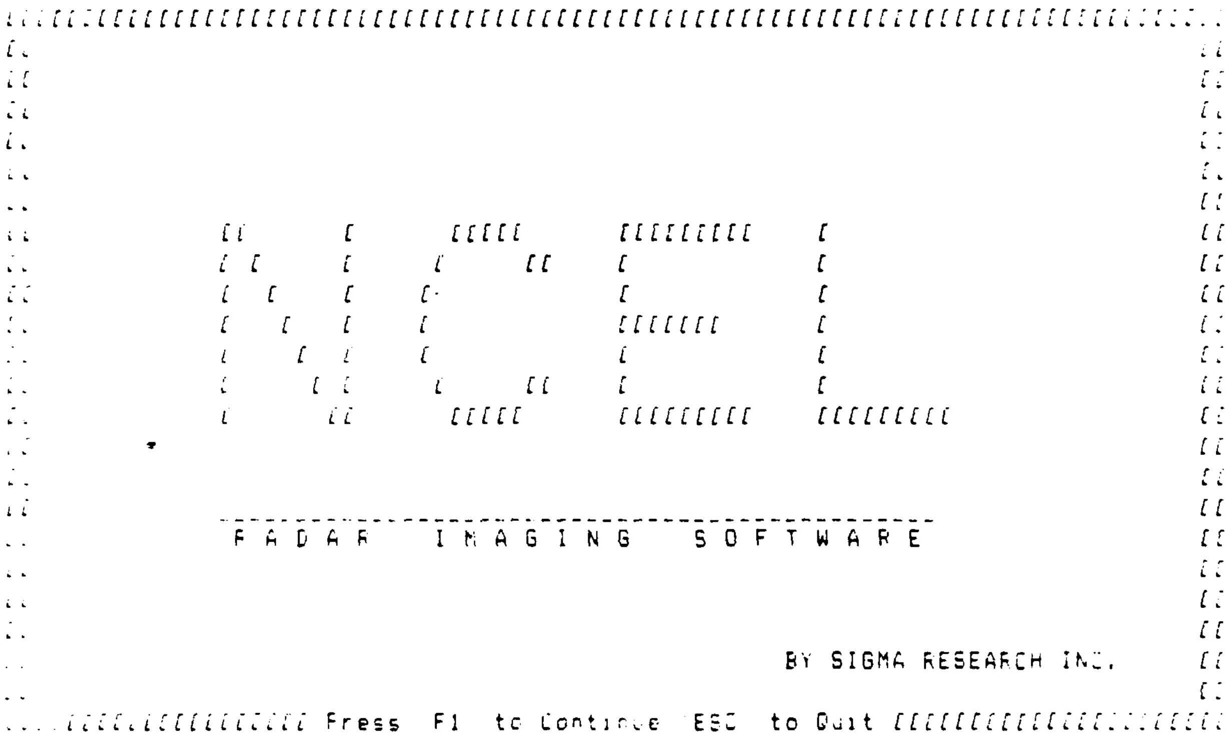
PREVIEW is the first program on the selection menu. The PREVIEW menu is shown in Figure 3-4. The inputs to the program are 1) a byte formatted survey radar data file as specified in the appendices, and 2) the data parameters displayed on the menu. The outputs are: 1) A screen intensity plot of the radar data, a sample of which is shown in Figure 3-5, and 2) a data header file which contains the parameters that are displayed in the PREVIEW menu. This header file is appended to the front of the radar data by TRANSFER to define it for T-SAFT.

3.5.1 Menu Entries

1. Operator: A name displayed on the menu and not used in the program.

2. Input filename: The name of the input raw survey radar file. The full name including path and drive must be specified. Note: When TRANSFER is executed it expects the raw data file to be in the same drive and path specified here. Valid names are consistent with the PCDOS requirement; eight characters long with a three character extension.

3. Output file name: This is the name that will be assigned to the output files, the header and the screen plot. File drives and paths are used to place these files in the specific drive and directory. Required disk space for these files is 128 bytes and 16384 bytes, respectively. Sufficient space must be provided otherwise the program will exit back to DOS with an error message.



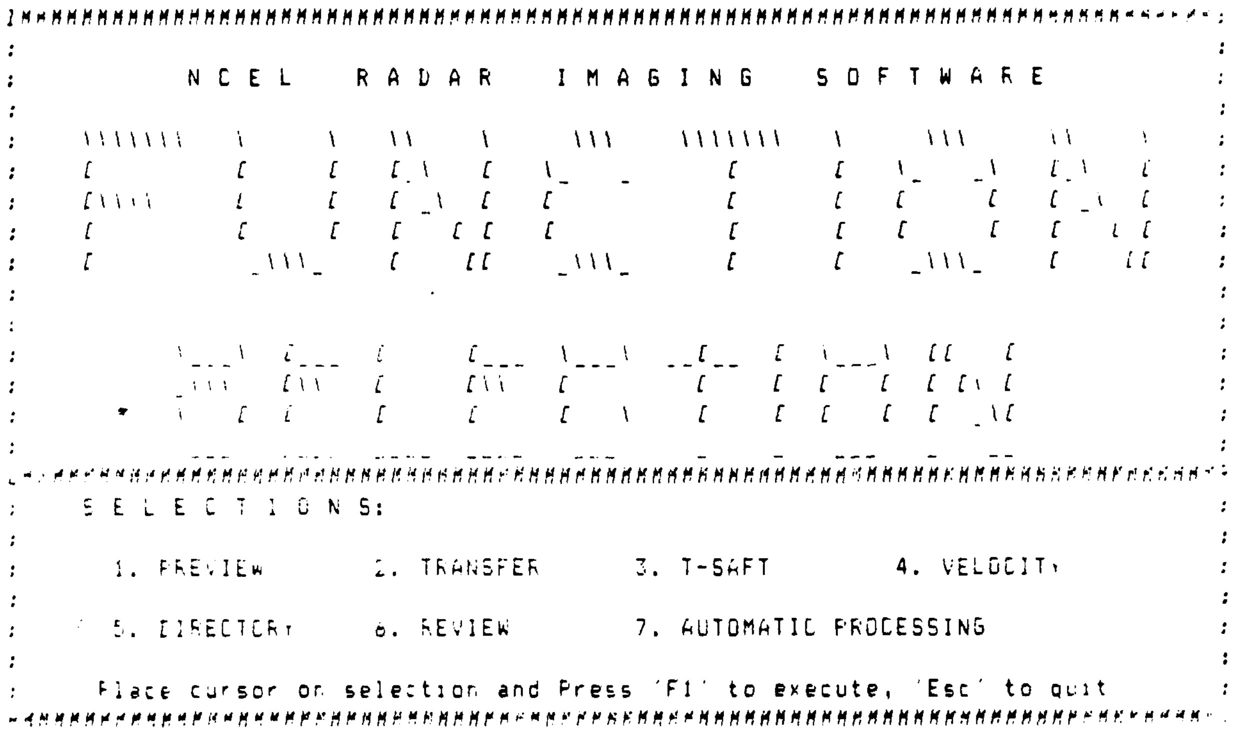
NCEL

RADAR IMAGING SOFTWARE

BY SIGMA RESEARCH INC.

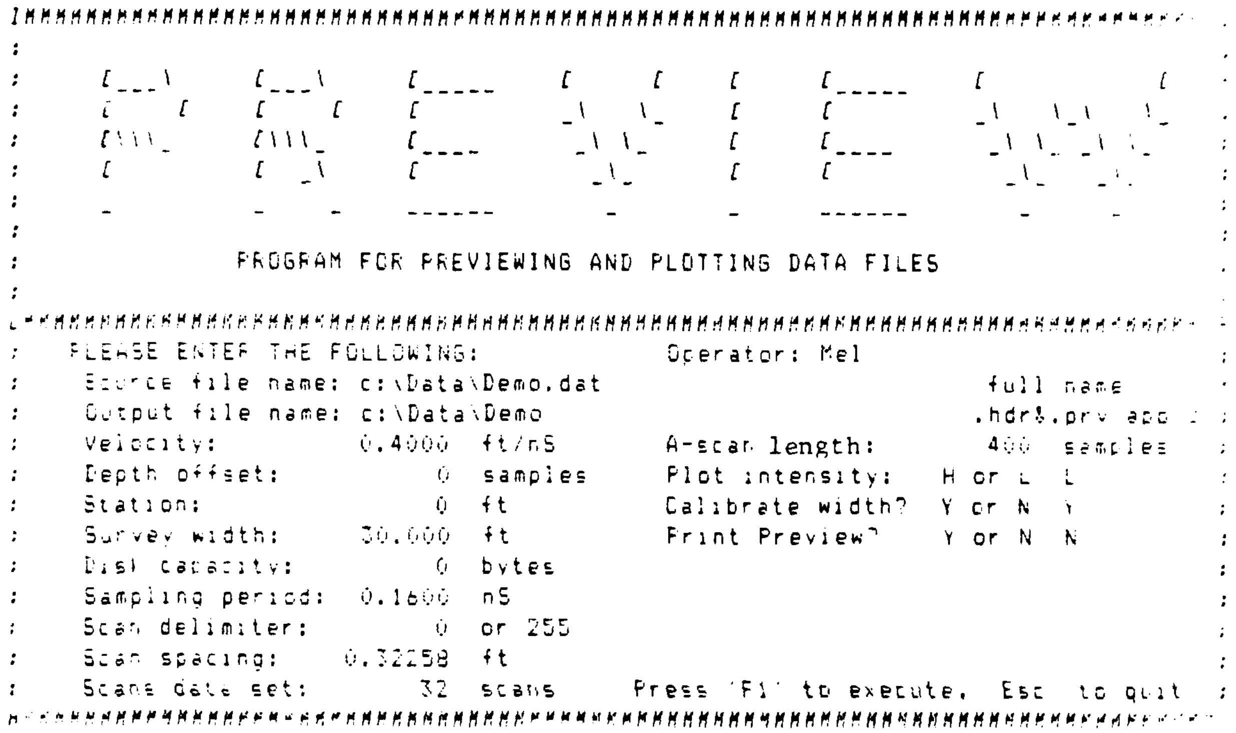
Press F1 to Continue ESC to Quit

Figure 3-2. Title screen.



```
N C E L   R A D A R   I M A G I N G   S O F T W A R E

S E L E C T I O N S:

  1. PREVIEW       2. TRANSFER       3. T-SAFT       4. VELOCITY

  5. DIRECTORY     6. REVIEW         7. AUTOMATIC PROCESSING

  Place cursor on selection and Press 'F1' to execute, 'Esc' to quit
```

Figure 3-3. Function selection menu.



```
PROGRAM FOR PREVIEWING AND PLOTTING DATA FILES

PLEASE ENTER THE FOLLOWING:                  Operator: Mel
 Source file name: c:\Data\Demo.dat                              full name
 Output file name: c:\Data\Demo                                  .hdr&.prv app
 Velocity:           0.4000  ft/nS           A-scan length:       400  samples
 Depth offset:            0  samples         Plot intensity:   H or L  L
 Station:                 0  ft              Calibrate width?  Y or N  Y
 Survey width:       30.000  ft              Print Preview?    Y or N  N
 Disk capacity:           0  bytes
 Sampling period:    0.1600  nS
 Scan delimiter:          0  or 255
 Scan spacing:      0.32258  ft
 Scans data set:         32  scans      Press 'F1' to execute,  Esc  to quit
```

Figure 3-4. PREVIEW menu.

4. Velocity: This is the velocity of the radar propagation specified in feet per nanosecond. The range of permissible input is between 0 and 1. Velocity values entered here may be updated by the VELOCITY program and the REVIEW program.

5. Depth offset: This parameter is used to correct the time shift in the recorded radar data. Each unit for this parameter represents one digitized sample. The value is obtained during field set up and recording of the radar data. Alternately, it can be estimated using the velocity determination routine. This number represents the relative position of the ground surface with respect to the start of data traces. For example, if the ground surface coincides with the start of the data trace then the depth offset is zero. In the data shown in Fig. 3-12 the ground surface as determined by the velocity determination routine is twenty samples after the start of trace (top of figure). This results in a +20 entry for depth offset. If the start of the recorded traces occurs after the ground surface, i.e. the ground surface has disappeared beyond the top of Fig. 3-12, then the depth offset would be a negative number.

6. Station: Station is the linear footage marking in the survey data. The number entered is in feet and specifies the footage at the left edge (beginning) of data. A positive entry is used when the survey is done towards increasing footage. A negative number is used when survey is done from high to low footage marking.

7. Survey length: This parameter is the total horizontal distance in feet traversed by the radar antenna in the particular raw data file. Example: If the antenna is pulled thirty feet, the digitized data represents a length of thirty feet.

8. Disk capacity: Disk capacity is a parameter that controls the display of the raw data on the screen. A zero or any number smaller than the length of the data file will result in a full screen plot of the data regardless of original file length. Any entry larger than the file length will proportion the plot on the screen according to the ratio of the file length and the data capacity. This feature is included so that preview plots can be directly compared while survey length may vary.

9. Sampling period: Sampling period is the time between two digitized samples of a radar trace. The value is obtained from the digitized radar system calibration signal. This sampling period is equal to the calibration signal period (normally 10 ns) divided by the number of digitization points in that one calibration signal period.

10. Scan delimiter: The scan delimiter is a marker that identifies the beginning of each radar trace (A-scan). It is a number selected by the digitizer, normally a 0 or 255 in an eight bit digitizer.

11. Scan spacing: Scan spacing is the horizontal distance between two radar traces. This is an approximate number and will be recalculated when the program is executed.

12. Scans/data set: This variable is the number of radar traces to process for each T-SAFT image reconstruction. The range for this value is 2 to 32. It is recommended that the default value of 32 be used for all processing. A smaller number will give a coarser image.

13. A-scan length: This is the number of digitized samples per radar trace. This is defined by the digitizing of the raw analog data. The range is between 1 and 400. The default value of 400 is recommended for higher resolution for all processing.

14. Plot intensity: Select 'H' for high intensity and 'L' for low intensity of the preview plot.

15. Calibrate width: 'Yes' is the default and is selected when station annotation on the preview plot is desired. When calibrate is selected the operator can calibrate the data length according to bench marks in the data. The survey length and the station entry is over written when the calibration is completed. When no length calibration is selected, the menu entered survey length and station are used for the data and no station annotation will be shown on the preview plot.

16. Print preview: The yes option will print the preview plot on the dot matrix printer at the conclusion of the program execution. In any case the plot would be written into a file with the extension 'PRV'.

3.5.2 PREVIEW Execution

When the menu entries have been updated, PREVIEW is executed by pressing 'F1'. The program control and the menu parameters are passed to a working program called 'PREVIEWK'.

Two possible modes of operation are executed depending on the calibration width option selected. If no width calibration is selected then PREVIEWK will read and plot the data. When it is done reading and plotting all the data it will write the header and preview plot file with the 'hdr' and 'prv' extension to the output file name. No output file is written if the output filename is blank. Printing of the display will be carried out if the option is invoked. The program control then returns to the preview menu program and the menu is displayed. Figure 3-5 shows a sample radar data plot generated by PREVIEW.

3.5.2.1 Calibrate Width

When the calibrate width option is invoked, the process is the same as above with the following additions. After reading and plotting the data, the program prompts the operator to move a diamond cursor to a left marking towards the left edge. Cursor movement is effected by the arrow keys on the keyboard. The direction of the arrows moves the cursor in the same directions. The shift key is used in conjunction with the arrow keys to increase the cursor movement speed.

At the left benchmark press 'F1'. Another prompt will ask that the cursor be moved to the right bench mark. When the cursor is on the right mark depress 'F2'. Then the prompt asks for the width in feet between the

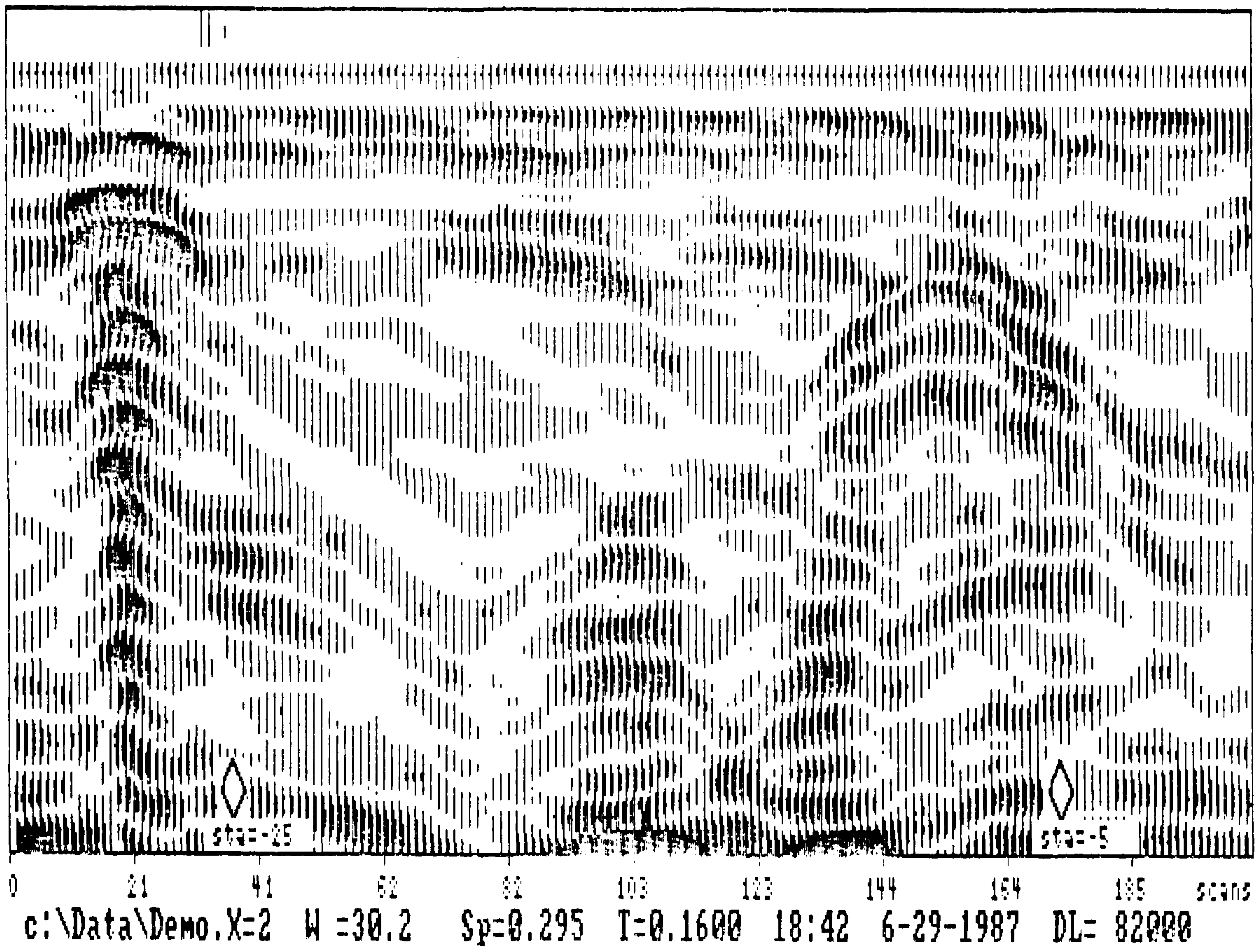


Figure 3-5. Sample PREVIEW plot of radar survey data taken at Hanford by Battelle Northwest.

marks. Enter width and depress <CR>. If a mistake is made, the steps can be repeated.

3.5.2.2 Station Entry

The station entry is begun after the width is calibrated. A prompt will appear asking the operator to move the cursor to the station mark. Pressing 'F3' will bring another prompt for a station number. As in the menu entry a positive integer number will label the horizontal axis in ascending values to the right and conversely a negative station number will be to the left. The station will be annotated on the plot with a diamond and number. This process can be repeated until 'F4' is pressed. The recorded station number is the last one entered.

3.5.2.3 Exit

Pressing 'F4' completes the calibration, printing and data saving is done as above, and control is returned to the PREVIEW menu.

3.6 TRANSFER

3.6.1 Menu Parameters

The transfer menu is shown in Figure 3-6. The inputs to this program are: 1) PREVIEW header file, 2) PREVIEW data plot, 3) survey raw data file, and 4) the menu parameters. The outputs are: 1) A T-SAFT data file with the data parameters appended in front of the data, and 2) a corresponding plot of the data in wiggle format. A sample is shown in Figure 3-8.

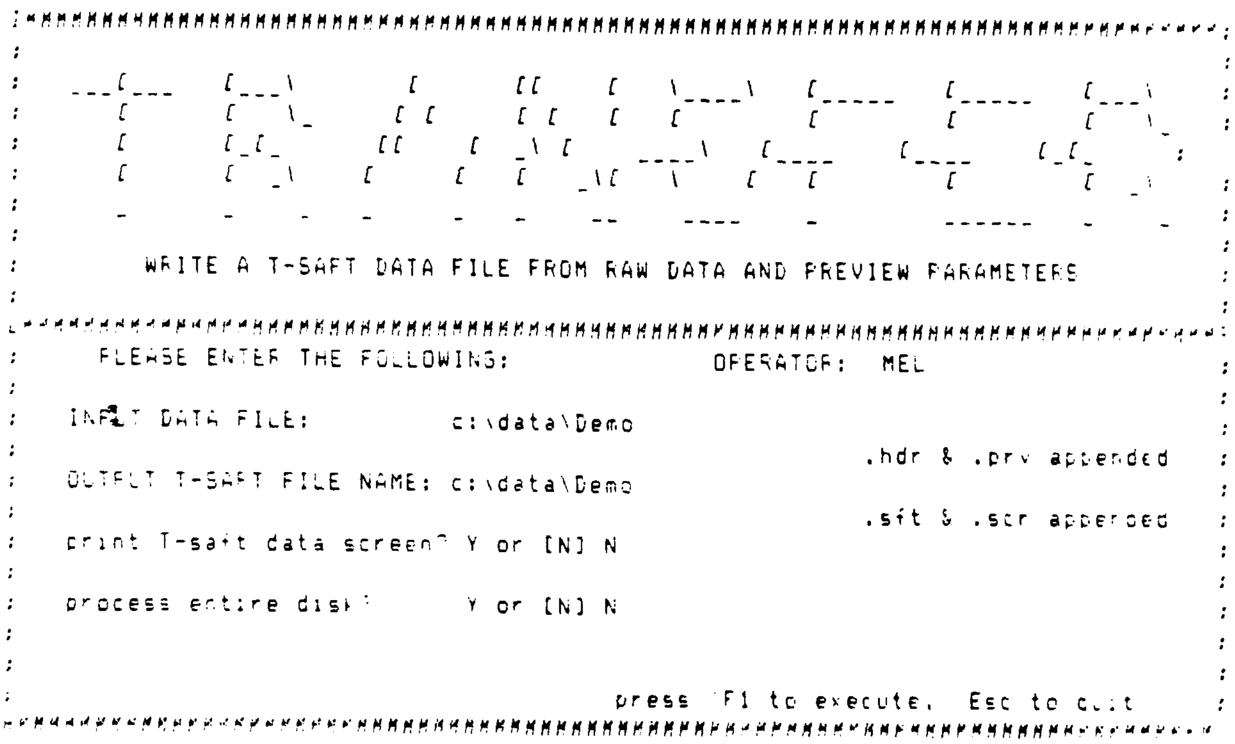
1. Operator: This is not used in the program.

2. Input data file: Enter the drive, the path, and root name of the PREVIEW output file to be used for selecting the data segments for T-SAFT. The header and the plot files generated by PREVIEW are used here. The header is appended to the front of the T-SAFT data file and the plot is used to select the data areas. The header contains the name of the raw radar data file.

3. Output T-SAFT file name: This name defines the drive, path, and root of the TRANSFER output files; '.sft' and '.scr' are appended to the name. The TRANSFER writes the '.sft' file for T-SAFT as input. The output file name should not contain any numbers if the T-SAFT automated processing feature is to be used on the data.

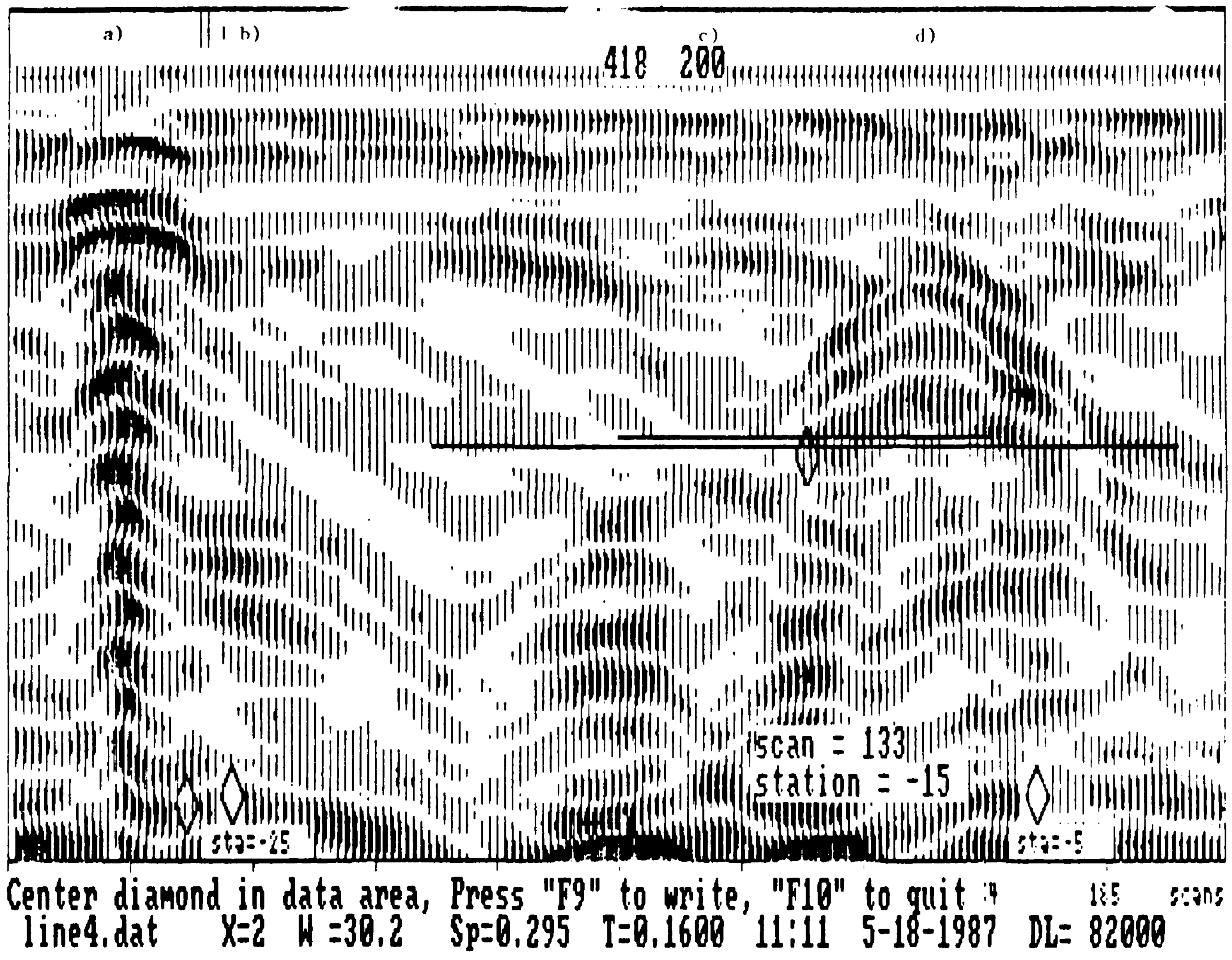
4. Print T-SAFT data screen?: As in PREVIEW, a 'yes' selection will cause the program to print the T-SAFT data screen when TRANSFER is done.

5. Process entire disk?: The default is the no option. TRANSFER will write only one T-SAFT input file of the manually selected data area. The output file names would be as described above. When the 'yes' option is selected, TRANSFER will automatically step through the full length of the raw data from the beginning until the data is exhausted, writing overlapping consecutive T-SAFT data files. The overlap is necessary so that when T-SAFT processes the data, the resultant image



```
        WRITE A T-SAFT DATA FILE FROM RAW DATA AND PREVIEW PARAMETERS

     PLEASE ENTER THE FOLLOWING:                 OPERATOR:  MEL

   INPUT DATA FILE:          c:\data\Demo
                                                   .hdr & .prv appended
   OUTPUT T-SAFT FILE NAME: c:\data\Demo
                                                   .sft & .scr appended
   print T-saft data screen? Y or [N] N

   process entire disk?      Y or [N] N


                                      press F1 to execute,  Esc to quit
```

Figure 3-6. TRANSFER menu.



```
Center diamond in data area, Press "F9" to write, "F10" to quit    185    scans
 line4.dat    X=2  W =30.2   Sp=0.295  T=0.1600  11:11  5-18-1987  DL= 82000
```

Figure 3-7. Data selection program (transfer) display of Battelle survey line 4 data. From left to right the targets in the display are a) surface metal plate, b) plastic pipe, c) plastic pipe, and d) metal pipe. Data selected for imaging is centered around diamond and represented by the long bar. The short bar shows the reconstructed image area.

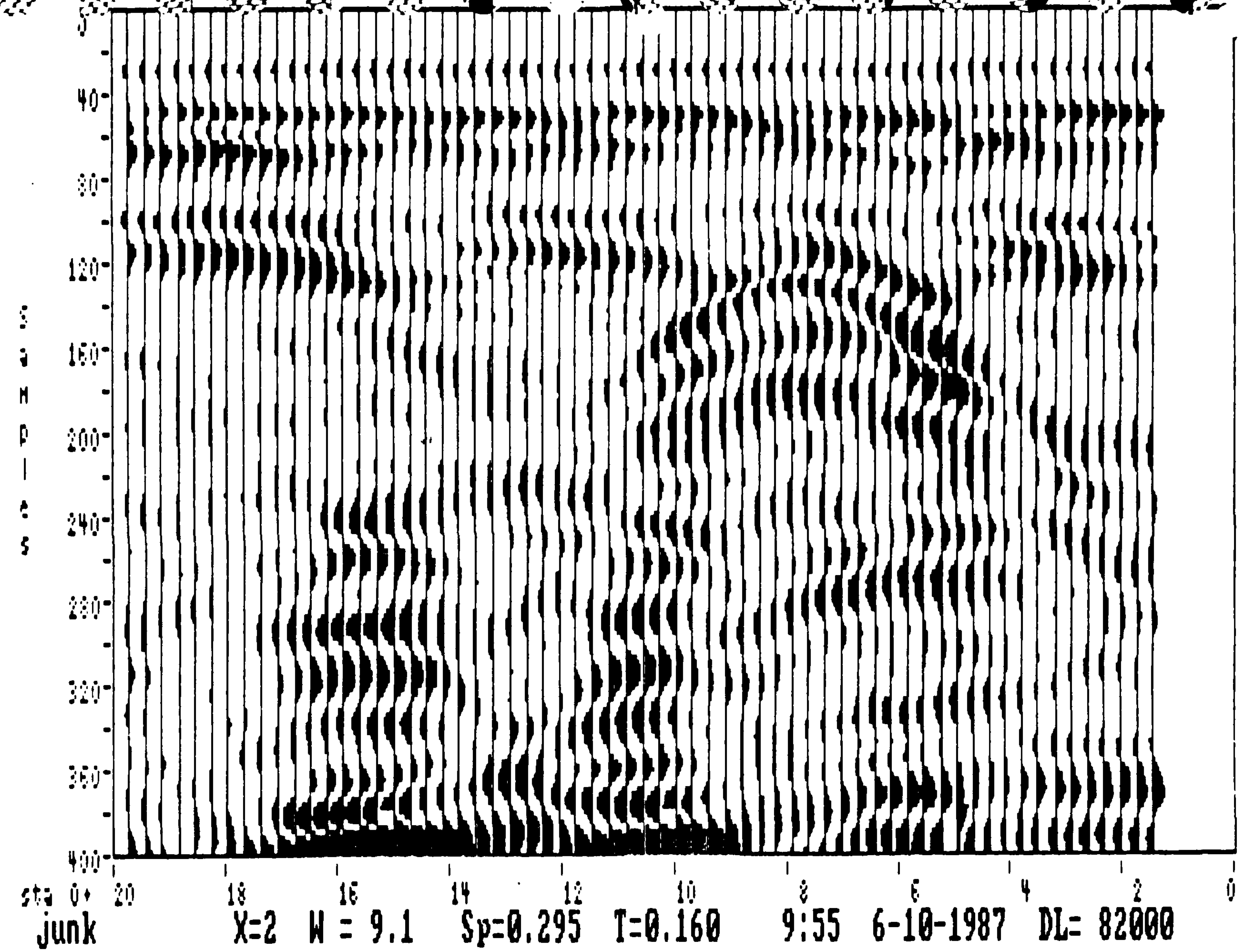


Figure 3-8. T-saft input data corresponding area spanned by long horizontal bar in Figure 3-7. This data is generated by data selection program called "transfer".

will be continuous. These files will have names derived from the entered output file name in the indexes starting from 0 and going up to 98. These indices are appended to the end of the file name if the so constructed file name is less than eight characters long, otherwise the last character of the name is dropped to make room for the index. Each of the T-SAFT files would represent approximately 10 ft. or less in reconstructed survey width.

3.6.2 TRANSFER Execution

The execution of TRANSFER is started by filling and selecting the desired parameters above. When the cursor is on the input file name depressing <CR> will initiate a check for the existence of the input file and the raw data file coded in the header that is specified by the input file name. If both files are found then a confirmation message is given, otherwise, an error message is displayed. Depressing 'F1' also performs this file checking.

When the files exist, depressing 'F1' transfer control to TRANSFWK, the working program. TRANSFWK displays the preview data plot and a horizontal compound cursor (see Fig. 3-7). The diamond is the center of the cursor. The short bar represents the reconstructed image width and the long bar spans the data traces required for T-SAFT. The length of the bars are automatically scaled with the varying scan spacing. Data selected beyond the available data will be filled with zeros.

3.6.2.1 Single Process Mode

In the single manual data selection processing (process entire disk option = NO) the compound cursor is manually moved to the area of interest and the 'F9' key is depressed to begin T-SAFT data writing. Station number is automatically recalculated for that data set. A plot of the data written is displayed and written into disk with the '.scr' extension. A printout of the T-SAFT data screen is generated if that option is selected (see Fig. 3-8). After processing one T-SAFT data file TRANSFWK displays a prompt for the next desired action. To quit and go back to the TRANSFER menu press 'F10'. Press 'F8' if one wishes to select and write another T-SAFT data file from the current raw data. When 'F8' is pressed the prompt asks for the next output file name. Upon entry of the next file name the PREVIEW screen is redisplayed with its compound cursor. Another area can be selected and the process can be repeated as needed.

3.6.2.2 Multi-Process Mode

In the automatic mode (process entire disk? option = 'Yes') the program TRANSFWK sequences through the following steps. It automatically indexes the file name, writes the data, and saves the output plot on disk. Then the output plot is printed on the printer if this option is selected. The above sequence is repeated until the data is exhausted.

When the data processing is completed in either the automatic or manual selection mode, the program control is returned to the TRANSFER menu program.

Caution: The output T-SAFT data file is 56,576 bytes long and the screen plot is 16,384 bytes long. The operator has to make sure that space is available to store all the output files on the destination drive or the program will halt and return control to DOS.

3.7 T-SAFT

The T-SAFT menu is shown in Figure 3-9. The inputs to this program are: 1) The transfer generated T-SAFT input file, and 2) the menu parameters. The output is a reconstructed image of the underground target as depicted in Figure 3-10.

3.7.1 Menu Parameters

The menu parameters are shown in Figure 3-9 together with the default values. A number of the parameters are identical and serve the same as those functions listed in the PREVIEW menu. Previously entered parameters can be recalled from the data header to update the default values. More discussion on this feature is found in 3.7.2. The parameters that need further clarification are listed below.

1. Data file name: This name should identify an existing T-SAFT data file name previously generated by TRANSFER. This name will also be used by T-SAFT for naming the output reconstruction image. The '.sft' will be replaced by a '.img'.

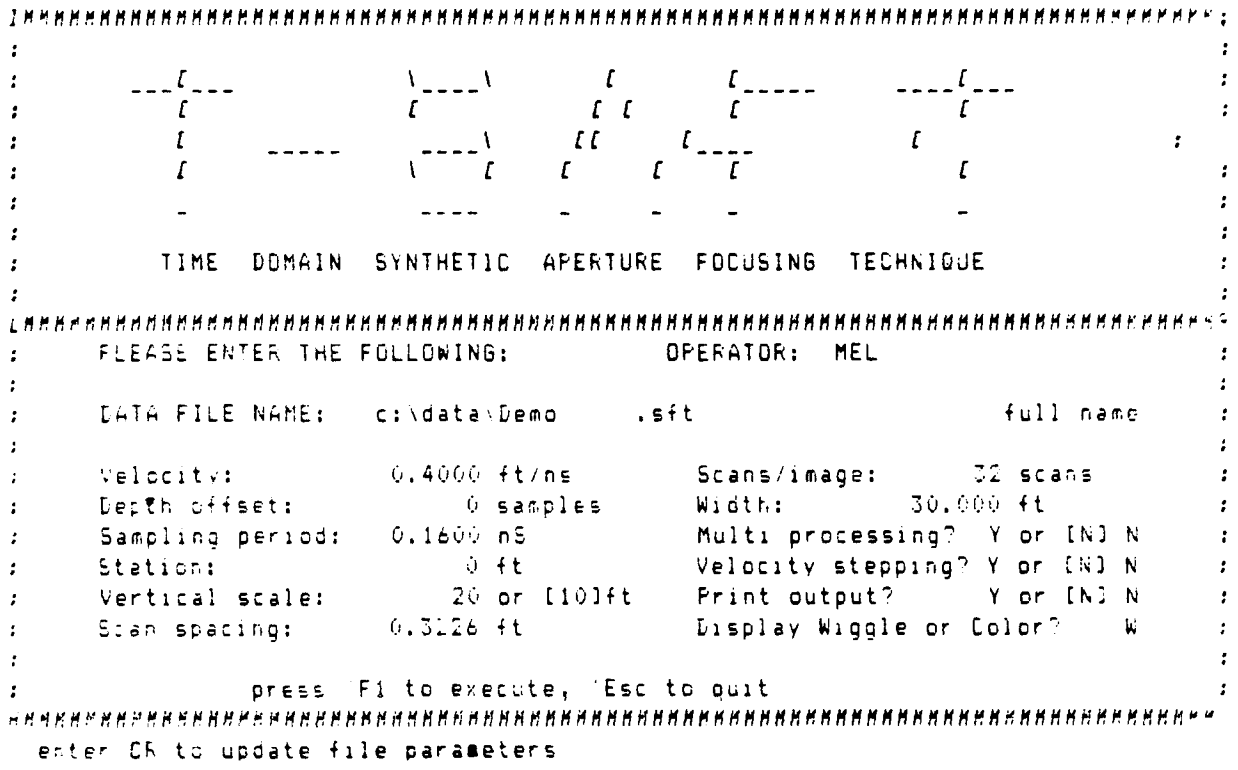
2. Vertical scale: The default value of the vertical scale is 20 ft. It is the vertical scale of the T-SAFT reconstructed output image. If more detail is required and the depth beyond 10 ft. is not of interest a scale change to 10 ft. will double the depth resolution.

3. Multiprocessing?: The multiprocessing parameter here is analoguous to the automatic processing of the entire disk in TRANSFER. With a 'yes' selection T-SAFT will process sequentially indexed T-SAFT data files starting from the current file specified. When the next file to be processed is not found a message is written and the processing terminates. If this option is not selected T-SAFT will process the currently specified file and stop at the menu.

4. Velocity stepping?: A 'yes' option would step the velocity through five values centered around the specified velocity. Each step is five percent of the specified velocity. The five images reconstructed will be assigned names with '.im0' to '.im4'. This is a useful feature when velocity tuning is needed. If a 'no' option is selected T-SAFT reconstructs the image only once at the specified velocity.

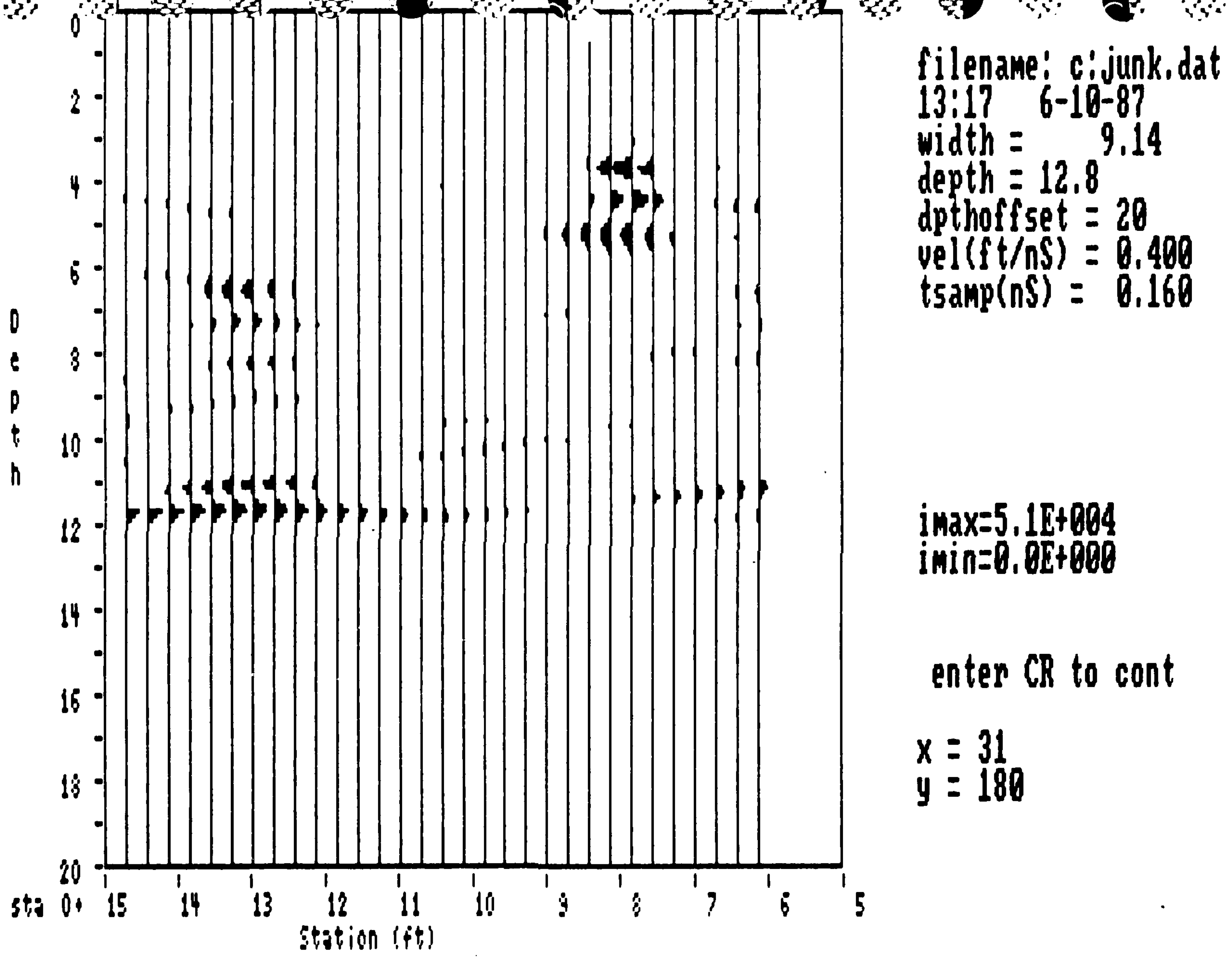
5. Print output?: A 'yes' will invoke the printing routine after each reconstruction. A 'no' selection produces no paper copy. In both cases the output image is stored on disk.

6. Display wiggle or color?: The two valid options are 'W' and 'C'. If 'W' is selected, the output image is plotted in the filled-in bipolar wiggle format. The polarity of the signal is preserved from the raw data. A wiggle to the right is positive while a wiggle to the left



```
IMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM;
:                                                                                  :
:      ___[___          \____\      [       [_____      ____[___                   :
:         [             [          [ [      [               [                      :
:         [     _____    ____\    [[     [____            [                   :
:         [             \    [   [     [   [                [                      :
:                                                                                  :
:         -              ----    -     -   -                -                      :
:                                                                                  :
:        TIME  DOMAIN  SYNTHETIC  APERTURE  FOCUSING  TECHNIQUE                    :
:                                                                                  :
LMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM9
:     PLEASE ENTER THE FOLLOWING:          OPERATOR:  MEL                          :
:                                                                                  :
:     DATA FILE NAME:   c:\data\Demo     .sft                        full name     :
:                                                                                  :
:     Velocity:            0.4000 ft/ns        Scans/image:       32 scans         :
:     Depth offset:             0 samples      Width:         30.000 ft            :
:     Sampling period:     0.1600 nS           Multi processing?  Y or [N] N       :
:     Station:                  0 ft           Velocity stepping? Y or [N] N       :
:     Vertical scale:          20 or [10]ft    Print output?      Y or [N] N       :
:     Scan spacing:        0.3126 ft           Display Wiggle or Color?    W       :
:                                                                                  :
:               press  F1 to execute, Esc to quit                                  :
MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM
  enter CR to update file parameters
```

Figure 3-9. T-SAFT menu.



```
filename: c:junk.dat
13:17   6-10-87
width =     9.14
depth = 12.8
dpthoffset = 20
vel(ft/nS) = 0.400
tsamp(nS) =  0.160


imax=5.1E+004
imin=0.0E+000


 enter CR to cont

x = 31
y = 180
```

Figure 3-10. T-saft reconstructed image of the survey line 4 around the selection diamond in Figure 1. The line target indicated as c) and d) are reconstructed here at station 13 and 8. From the survey notes, target at station 13 is a plastic pipe and target at station 8 is a metal pipe. Note the weak plastic target signal has been very successfully focussed in this image.

means the data was negative. At best, the wiggle format displays 12 magnitude steps. At worse it displays six grey levels. If 'C' is selected, the display is a color display of three colors plus black. This color display can be printed but not stored on disk.

3.7.2 T-SAFT Execution

T-SAFT execution is started by depressing 'F1' after the parameters are entered. A feature has been implemented to update the default values with the values stored in the header file of the T-SAFT data file. This is done by pressing <CR> when the cursor is on the data file name. These updated parameters can again be changed if needed by entering new values. If the name does not specify a valid file an error message is issued, otherwise a proceed message is given.

When the T-SAFT program is executed it reads in the data and performs the image reconstruction as outlined in Appendix A. During processing iteration numbers starting from 0 ascending to 399 are displayed on the screen. As the processing is completed, the reconstructed image is drawn on the screen and printed if the option was selected. If no printing is selected, the program waits for a <CR> after displaying the image. When the <CR> has been entered, the program writes the screen display to file with a '.img' extension and returns to the T-SAFT menu.

3.7.2.1 Multiprocessing

In the multiprocessing mode the above processing is repeated over and the file name index is incremented by one each time. Processing stops at the menu when no more files are found or when the index exceeds 98. There is no waiting for the <CR> to be depressed between image processing in multiprocessing.

3.7.2.2 Velocity Stepping

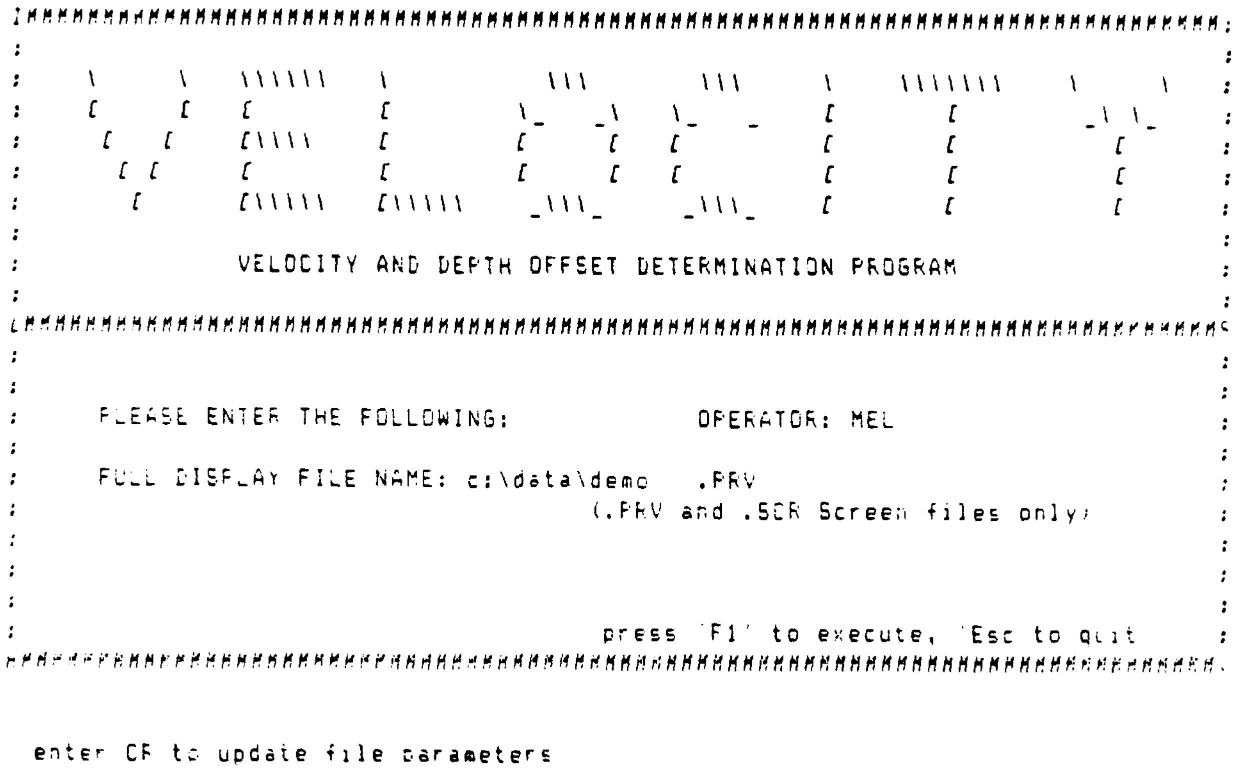
If the velocity stepping is invoked each input file is processed five times with five velocity steps each 5% apart centered about the specified velocity. This algorithm applies to both single file processing and multiprocessing.

The annotation on the image display shows the names of the T-SAFT input and output files, the time and data of the image reconstruction, the velocity, depth offset, total data depth, and image intensity. The vertical axis is scaled in feet and the horizontal axis is marked with the station numbers rounded to the nearest foot.

Caution: Again the operator must make sure that there is enough space for the output image file which is 16,384 bytes each.

3.8 VELOCITY

The menu for this program is displayed in Figure 3-11. The input to this program is either a PREVIEW or TRANSFER generate T-SAFT plot. The outputs are the velocity and depth offset values which can be automatically updated in the data header files.



```
:                                                                              :
:     \      \    \\\\\\    \          \\\       \\\      \     \\\\\\\    \     \  :
:     [      [    [         [        \_    _\   \_    _    [       [        _\ \_  :
:      [   [      [\\\\     [        [      [   [          [       [          [     :
:       [ [       [         [        [      [   [          [       [          [     :
:        [        [\\\\\\   [\\\\\\   _\\\_      _\\\_     [       [          [     :
:                                                                              :
:               VELOCITY AND DEPTH OFFSET DETERMINATION PROGRAM                :
:                                                                              :

:                                                                              :
:                                                                              :
:     PLEASE ENTER THE FOLLOWING:             OPERATOR: MEL                    :
:                                                                              :
:     FULL DISPLAY FILE NAME: c:\data\demo    .PRV                             :
:                                      (.PRV and .SCR Screen files only)       :
:                                                                              :
:                                                                              :
:                                                                              :
:                                       press 'F1' to execute, 'Esc to quit    :


  enter CR to update file parameters
```

Figure 3-11. VELOCITY menu.

3.8.1 Menu Parameters

The menu parameters for this program consist of only two entries.

1. Operator: This entry is not used by the program.

2. Display file name: This is the input file name for VELOCITY. It specifies a screen plot that has been previously generated by PREVIEW or TRANSFER.

3.8.2 VELOCITY Execution

Depressing 'F1' starts the execution by checking for the existence of the file. If found, program control is transferred to VELOCIWK, the velocity determination working program. If the file is not found the program stays in the menu and prompts an error. Once in the VELOCIWK program the screen plot is loaded and displayed on the CRT. The next operation is to calibrate the width of the displayed data as in PREVIEW. A prompt asks the operator to move an inverted 'V' to a left reference and depress 'F1'. Then the program will ask that the 'V' be moved to a right reference and depress 'F2'. When this is done a prompt asks for the distance in feet between these reference points. Next the sampling period is requested. The sampling period is the time represented by each digitized sample. It is the same as that value in PREVIEW. As soon as this is entered VELOCIWK is ready to generate the template hyperbola.

3.8.2.1 Hyperbola Matching

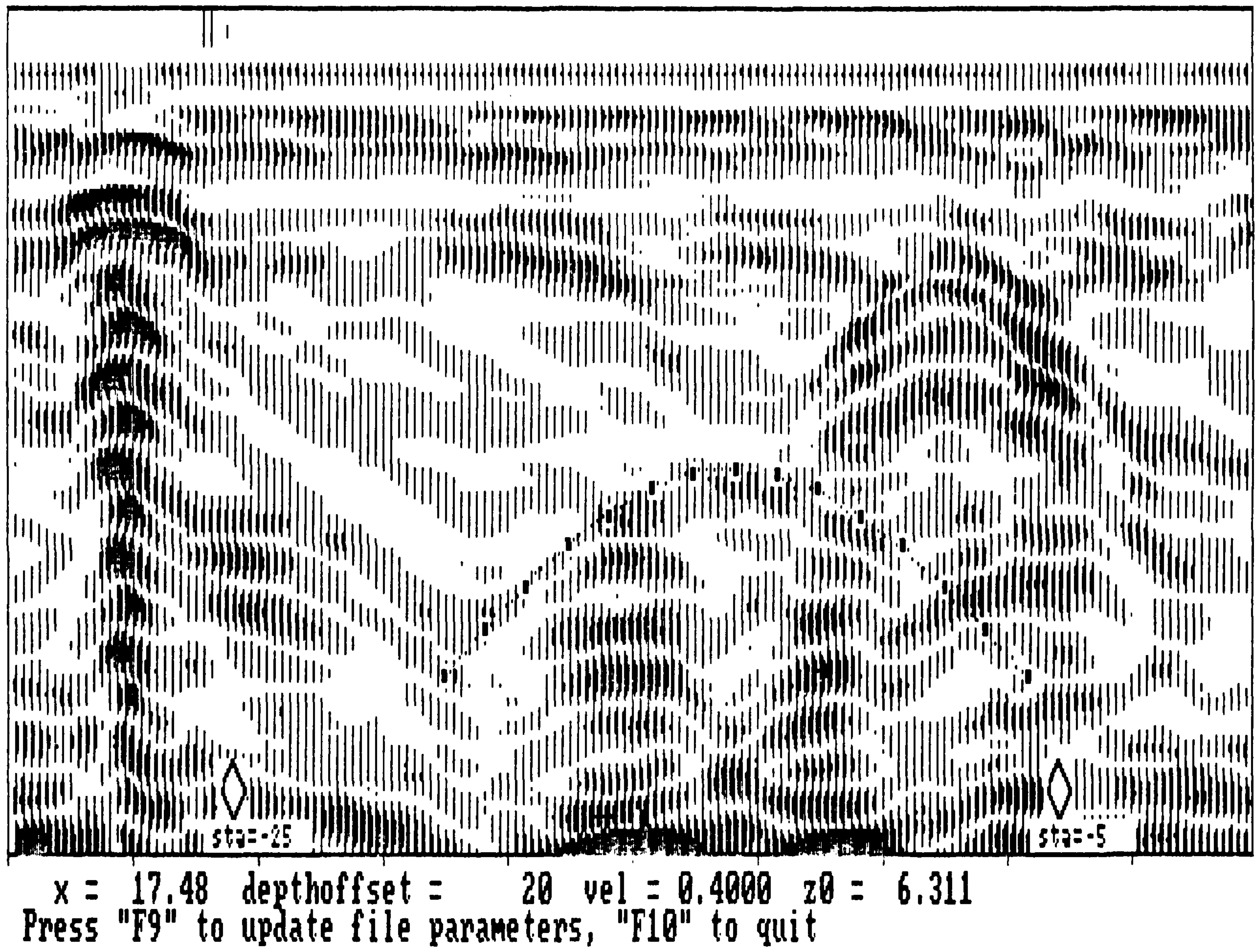
Figure 3-12 shows an example of the VELOCITY program in operation. The template hyperbola is matched to the hyperbola formed by the radar data (target hyperbola). This program works best when the wings of the target hyperbola extend some distance from the apex and have good signal strength. A radar data survey with multiple target nearby at different depth would allow the velocity matching to be cross checked using the different targets.

3.8.2.1.1 Template Motion and Shape Changing

The template is translated horizontally and vertically using the arrow keys. The direction of the arrows corresponds to the motion of the template. A template generated at zero depth assumes the shape of the asymptotes. The 'page up' and the 'page down' keys are used to increase and decrease the depth of the template. The velocity is increased and decreased using the home and 'end' keys. A default value of 0.4 feet per nano second is initially set. The shift key used in conjunction with any of these keys increases the rate of motion or shape change. The X coordinates, depth offset, velocity, and depth of the template apex are displayed at the bottom of the screen.

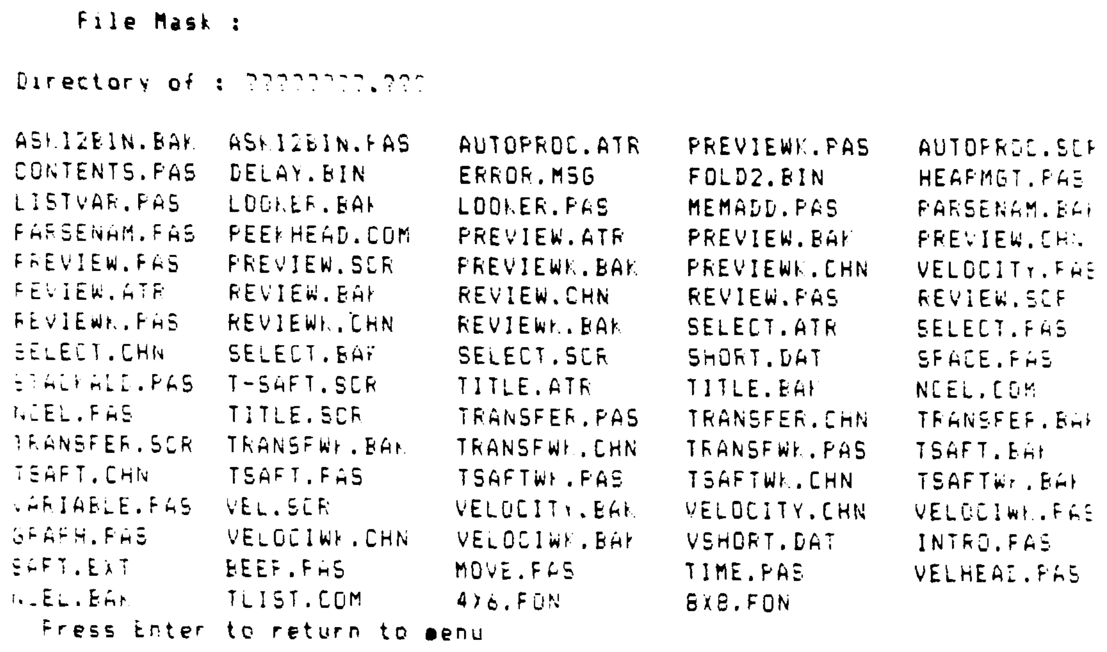
3.8.2.1.2 Procedure for Template Matching

1. If the depth offset is known, translate the template vertically until this offset reads correctly. A positive offset is used if the datum is to be shifted down from the beginning of trace. A negative offset shifts the datum up before the beginning of trace.



```
  x =  17.48  depthoffset =     20  vel = 0.4000  z0 =  6.311
Press "F9" to update file parameters, "F10" to quit
```

Figure 3-12. Sample template velocity curve fit on deeper target.



```
     File Mask :

Directory of : ????????.???

ASK12BIN.BAK   ASK12BIN.PAS    AUTOPROC.ATR    PREVIEWK.PAS    AUTOPROC.SCR
CONTENTS.PAS   DELAY.BIN       ERROR.MSG       FOLD2.BIN       HEAPMGT.PAS
LISTVAR.PAS    LOOKER.BAK      LOOKER.PAS      MEMADD.PAS      PARSENAM.BAK
PARSENAM.PAS   PEEKHEAD.COM    PREVIEW.ATR     PREVIEW.BAK     PREVIEW.CHN
PREVIEW.PAS    PREVIEW.SCR     PREVIEWK.BAK    PREVIEWK.CHN    VELOCITY.PAS
REVIEW.ATR     REVIEW.BAK      REVIEW.CHN      REVIEW.PAS      REVIEW.SCR
REVIEWK.PAS    REVIEWK.CHN     REVIEWK.BAK     SELECT.ATR      SELECT.PAS
SELECT.CHN     SELECT.BAK      SELECT.SCR      SHORT.DAT       SPACE.PAS
STACKALC.PAS   T-SAFT.SCR      TITLE.ATR       TITLE.BAK       NCEL.COM
NCEL.PAS       TITLE.SCR       TRANSFER.PAS    TRANSFER.CHN    TRANSFER.BAK
TRANSFER.SCR   TRANSFWK.BAK    TRANSFWK.CHN    TRANSFWK.PAS    TSAFT.BAK
TSAFT.CHN      TSAFT.PAS       TSAFTWK.PAS     TSAFTWK.CHN     TSAFTWK.BAK
VARIABLE.PAS   VEL.SCR         VELOCITY.BAK    VELOCITY.CHN    VELOCIWK.PAS
GRAPH.PAS      VELOCIWK.CHN    VELOCIWK.BAK    VSHORT.DAT      INTRO.PAS
SAFT.EXT       BEEF.PAS        MOVE.PAS        TIME.PAS        VELHEAD.PAS
NCEL.BAK       TLIST.COM       4X6.FON         8X8.FON
  Press Enter to return to menu
```

Figure 3-13. Sample directory.

2. Translate the template horizontally to the vicinity of the target.

3. Move the template down in depth to straddle the target.

4. If the template is too narrow and the depth offset has been set correctly then increase the velocity and match the template again with the depth control. If the template is too wide then decrease the velocity and increase the depth. If the depth has not been set correctly, then make the best approximation with the depth and offset first before adjusting the velocity.

5. After a match has been made check the parameters by matching the template on another target of different depth by changing the horizontal location and depth only. If a match can be made in this manner then the velocity and offset values are correct, otherwise iterate between steps 4 and 5 until the best compromise can be made.

3.8.2.2 Updating the Velocity and Offset of a Header File

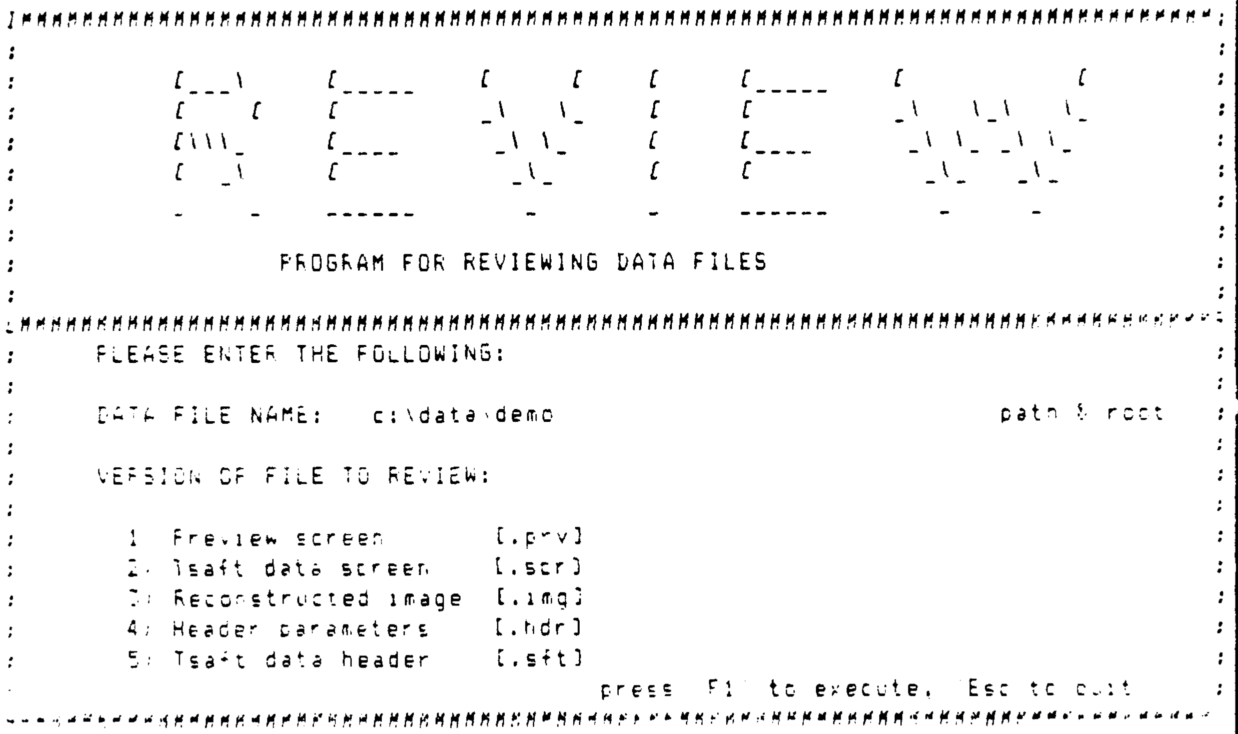
The updating can be done by pressing 'F9' and entering a complete file name of a file that contains a header. The PREVIEW header file or the T-SAFT input data file. More than one output file can be updated in this manner. Pressing 'F10' will exit VELOCIWK and the menu will be redisplayed. It is recommended that this velocity program be exercised on known data to gain a feel of its performance before a complete unknown is attempted.

3.9 Directory

3.9.1 Directory

The directory subroutine does not have a menu. It has only one entry, a file mask. The file mask is used for seeking and displaying file names in the specified directory. The mask can be up to eight characters for the name and three characters for the extension.

File mask examples	Filename displayed
*.PAS	Current directory files with .PAS extension.
XYZ?????.*	Current directory files starting with letters 'XYZ'.
.	All files in current directory
a:*.*	All files in drive a:
c: data *.*	All files in directory data in drive c:



```
PROGRAM FOR REVIEWING DATA FILES

PLEASE ENTER THE FOLLOWING:

DATA FILE NAME:   c:\data\demo                          path & root

VERSION OF FILE TO REVIEW:

  1  Preview screen          [.prv]
  2: Tsaft data screen       [.scr]
  3: Reconstructed image     [.img]
  4: Header parameters       [.hdr]
  5: Tsaft data header       [.sft]
                                   press  F1  to execute,  Esc to quit
```

Figure 3-14. REVIEW menu.

If no mask is specified, then all files in the current directory are shown. A sample directory with no file mask specified is shown in Fig. 3-13.

If a path is specified then the '*.*' or other mask must be specified.

3.9.2 Execution

Enter desired mask as outlined above, then press <CR>. A list of the file names will appear. Pressing <CR> again will display the selection menu.

3.10 REVIEW

The menu for REVIEW is shown in Figure 3-14. REVIEW is a program for looking at output files that have been generated by other programs. The input for this program is one of five possible versions of the intermediate screen plots or data files. The output is either a file print of the screen plot or a modified header.

3.10.1 Menu Entries

The only entry required is the file name.

1. Filename: Only the path and root portion of the filename is needed. The extension is automatically appended when the file version to review is selected. A file name entered with a file extension overrides the automatically appended extension. This feature is used to access files that have a name not conforming to the suggested extensions here.

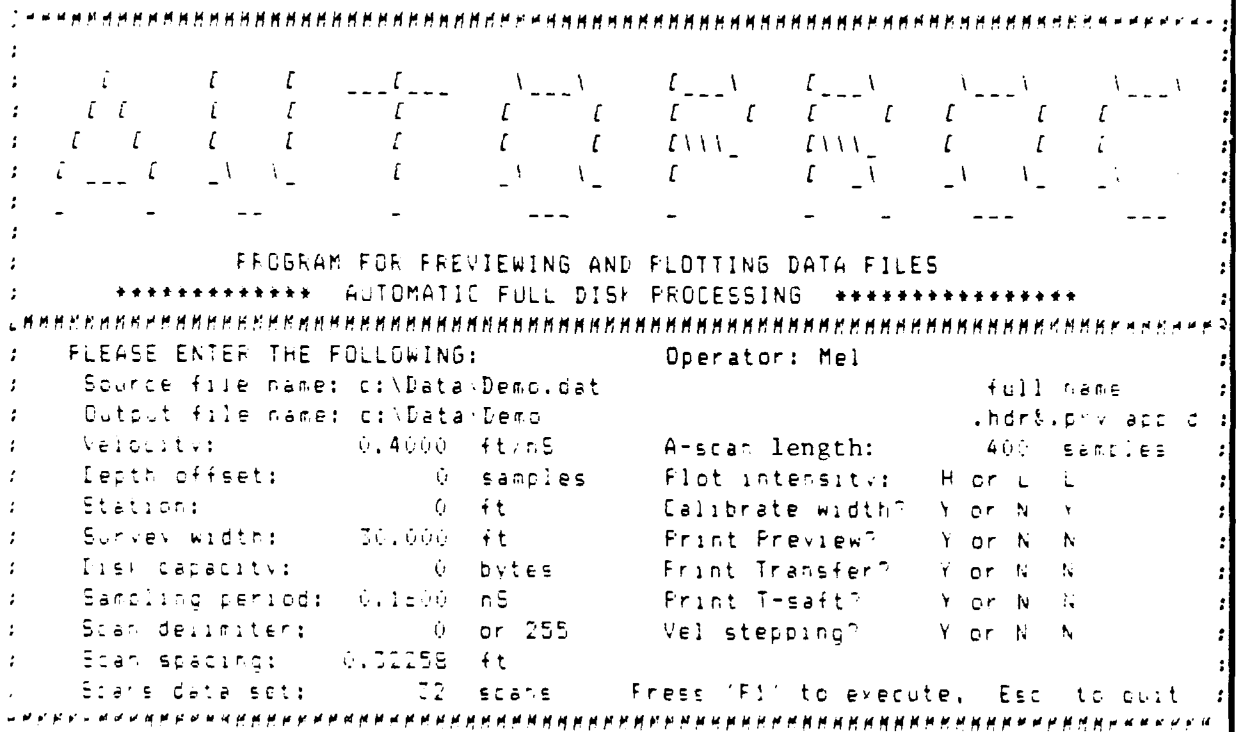
3.10.2 Execution

First enter filename, then move cursor to select the version of the file to review. Press 'F1' and REVIEWK will take over an display the content of the file. In the first three cases they are the screen files and they are displayed with a prompt to print or not to print. If the print option is selected the screen is printed otherwise the REVIEW menu is redisplayed when the <CR> is depressed.

With choices 4 and 5 the parameter headers are displayed as a list. A prompt asks whether any modification is desired on the values. If 'no' is selected the REVIEW menu is redisplayed. When a 'yes' is selected REVIEWK will relist each entry in turn and wait for a new input for that parameter. If no change is needed, enter <CR>. Otherwise, the new value, then enter <CR>. At the end of the list, control is returned to the REVIEW menu program.

3.11 AUTOPROC

The auto processing program is the most comprehensive of all these programs in terms of all these programs explained. It is actually a superset of the data processing programs namely PREVIEW, TRANSFER, and T-SAFT executing iteratively until data is exhausted. The menu for AUTOPROC is displayed in Figure 3-15. The program execution flow is shown in Figure 2-3.



```
         PROGRAM FOR PREVIEWING AND PLOTTING DATA FILES
   *************  AUTOMATIC FULL DISK PROCESSING  ****************

  PLEASE ENTER THE FOLLOWING:                  Operator: Mel
   Source file name: c:\Data\Demo.dat                          full name
   Output file name: c:\Data\Demo                          .hdr&.prv app d
   Velocity:              0.4000  ft/nS       A-scan length:       400  samples
   Depth offset:               0  samples     Plot intensity:   H or L  L
   Station:                    0  ft          Calibrate width?  Y or N  Y
   Survey width:          30.000  ft          Print Preview?    Y or N  N
   Disk capacity:              0  bytes       Print Transfer?   Y or N  N
   Sampling period:       0.1500  nS          Print T-saft?     Y or N  N
   Scan delimiter:             0  or 255      Vel stepping?     Y or N  N
   Scan spacing:         0.02258  ft
   Scans data set:            32  scans    Press 'F1' to execute, Esc  to quit
```

Figure 3-15. AUTOPROC menu.

3.11.1 Menu Entries

The menu entries of AUTOPROC are identical to those in the PREVIEW menu with three exceptions. There are additional entries; the print transfer, print T-SAFT, and velocity stepping. All these entries serve the same functions as in their respective menus. Please see velocity stepping in T-SAFT and the entries for the PREVIEW menu for further details.

3.11.2 Execution

As in all the preceeding functions the menu entries are first displayed with their default values. These parameters have to be updated by the operator to reflect the conditions of the field survey for the particular data. When 'F1' is depressed, the existence of the source file is checked and the survey data read. Figure 2-3 shows the execution flow control of the autoprocessing.

The execution of AUTOPROC starts with PREVIEWK. PREVIEWK performs exactly the same functions as it normally does when executed from the PREVIEW menu. The calibrate width option is executed if selected. The only change here is that at the end of the PREVIEWK processing the program control is turned over to TRANSFWK instead of going back to the PREVIEW menu. Similarly TRANSFWK does its processing in the process entire disk mode. That is, it will start writing consecutive T-SAFT data files from the left most data towards the right most data until the data is exhausted. As these processes are run the screen will show the same displays that one normally sees when the programs are run individually.

When TRANSFER has written all the data and screen files it relinquishes control to T-SAFTWK. T-SAFTWK reconstructs the first data set and returns control to the T-SAFT menu which increments the data file name index and returns control to T-SAFTWK if the file is found. If the next file is not found, the execution stops with the T-SAFT menu displayed. This concludes the automatic processing of the data disk.

Printed outputs are made if the print selection on the AUTOPROC menu are set to 'Yes'. In any case, all the output files are saved. Velocity stepping can also be performed in AUTOPROC.

Caution: Care must be taken to ensure that there is sufficient space for the output files on disk otherwise the program will exit to DOS when disk space is depleted.

3.12 OPERATION EXAMPLE

The following paragraphs outline a step by step data reduction example. References will be made to the preceeding sections for details, display and function clarification.

3.12.1 Set-Up and Installation

Step 1 Turn on power to computer, monitor, and printer.

Step 2 Copy all NCEL GPR Image Processing Software files to directory C: GPR.

Step 3 Copy file Demo.Dat to C: Data.

Step 4 Copy file Autoexe.Bat to E:

Step 5 Reset computer by either using the CTRL-ALT-DEL keys or turn power off and wait one minute.

3.12.2 Example Data Description

This example will use a field survey data names Demo.dat included in the distribution diskette taken at Hanford, Washington by Battelle Northwest. The data represents 30 ft. of traverse. The targets in the surveyed path are shown in Figure 3-16. The direction of data is actually from right to left. From the field notes, the survey is made with a 300 MHz GSSI radar unit; the approximate velocity of radar propagation is .40 ft./ns. The digitized samples represent a duration of 0.16 ns and the ground surface appears at approximately the 23rd sample from the beginning of the scan. The total length of each A-scan trace is 400 samples.

3.12.3 Start Data Processing

Step 1 Turn on power to computer system.

Observation: Computer self-checking messages.

Step 2 Type 'CD GPR'. Type 'NCEL'.

Observation: The NCEL title screen will be displayed (see Figure 3-2).

Step 3 Press 'F1' to go to function selection screen.

Observation: Function selection menu (Figure 3-3) is displayed and cursor links at Preview selection.

3.12.4 Preview Execution

Step 1 Select Preview function by depressing 'F1' while the cursor is next to the function.

Observation: The Preview menu (Figure 3-4) is displayed.

Step 2 With the down arrow key, move cursor to depth offset entry field. Type '23' for 23 offset samples and depress 'Enter'.

Observation: The default zero for depth is changed to 23.

Note: The default entries on this menu, except for the depth offset, have been set up to process the data Demo.dat. No other changes need be made except for experimentation.

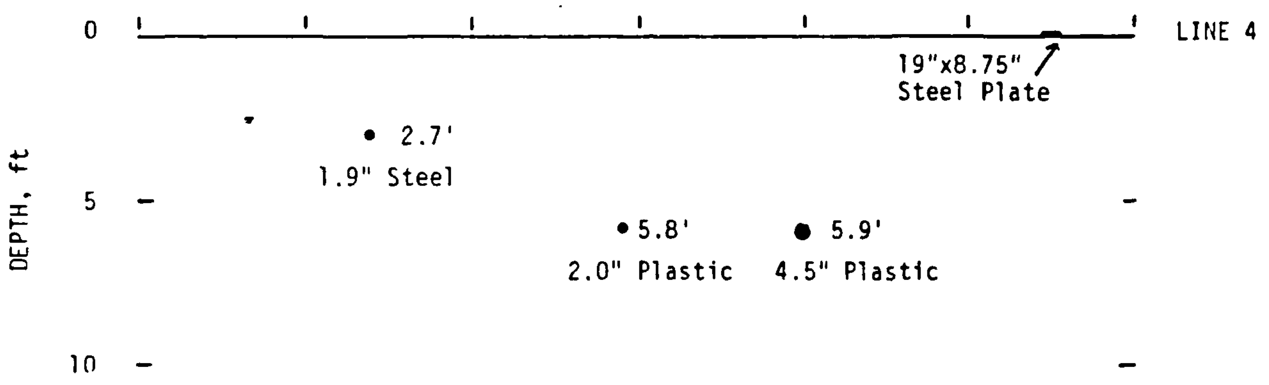


Figure 3-16. Target configuration of sample radar survey performed at Hanford, Washington by Battelle Northwest.

Step 3 Depress 'F1' to continue Preview execution.

Observation: The new data is plotted on the screen starting from left to right. The numbers along the bottom axis show the number of digitized traces. The numbers on the bottom right are the byte count of the current line being plotted and the total length of the data.

Step 4 When plotting is done, depress 'left arrow' to move diamond to left edge.

Step 5 Press 'F1' to mark this position.

Step 6 Move diamond to right edge using 'shift' and 'right arrow' keys.

Step 7 Depress 'F2' to mark right edge.

Step 8 Enter '30' for distance between the markings.

Observation: Prompt reads 'move diamond to marker then press 'F3'. Calspace = 0.2940'. Calspace is the calculated spacing between each A-scan.

Step 9 Move diamond with 'left arrow' to first visible marker (scan = 171) from the right.

Observation: A scan number corresponding to cursor position is visible in lower right quarter.

Step 10 Depress 'F3'.

Observation: 'Enter station [0ft]'. Zero is the default station. The position at scan =171 is 5 ft.. A negative number is to be entered because the data as displayed goes from high footage marking to low footage marking.

Step 11 Type '-5' and depress 'Enter' to add the station number.

Note: Any number of station annotation can be added but only the last one entered is used for calculations in the program.

Step 12 Depress 'F4' to conclude Preview.

Observation: Preview menu is redisplayed.

Note: Output files are written on disk. The files are as named on the Preview menu as Demo.hdr and Demo.prv. If the Print Preview option was selected then a print of the radar data plot would also be made.

Step 13 Depress 'ESC' to return to function selection menu.

Observation: Selection menu.

3.12.5 Transfer Execution

Step 1 Move cursor to select transfer with 'right arrow'.

Step 2 Depress 'F1' to invoke the Transfer menu.

Observation: Transfer menu as shown in Figure 3-6 is displayed.

Step 3 Move the cursor down to the input data file name field and depress 'Enter' to update the menu parameters.

Note: The input/output names are already correct for the demonstration data.

Step 4 Depress 'F1' to execute.

Observation: The radar data plot similar to the one in Figure 3-7 is displayed. The plot is the same as was previously generated by Preview.

Step 5 Move the compound cursor using the shift and arrow keys until scan = 133 as in Figure 3-7. The short bar represents the area of reconstructed image.

Step 6 Depress 'F9' to write this data into a T-SAFT compatible data format.

Observation: A new screen is displayed and the T-SAFT data is plotted in greater detail. The horizontal scale is the footage marking. This plot is shown in Figure 3-8.

Note: The data is written into the disk with the name C: data demo.sft. Prompt reads 'Press 'F10' to quit, 'F8' to write another data set'.

Step 7 Depress 'F10' to quit the Transfer processing.

Observation: Transfer menu is displayed.

Step 8 Depress 'ESC' to return to the function selection menu. Observation: The function selection menu is displayed.

3.12.6 T-SAFT Execution

Step 1 Select the T-SAFT function with the right arrow, press 'F1'.

Observation: The T-SAFT menu as in Figure 3-9 is displayed. The entries with the exception of the width and depth offset are correct for this data set.

Step 2 Move the cursor down with the 'down arrow' key to data file name.

Step 3 Depress 'Enter' to update the data parameter.

Observation: The station number and the width are updated per the data set parameters.

Step 4 Depress 'F1' to execute T-SAFT.

Observation: The screen clears and a message 'Reading data C: data demo.sft' is displayed, followed by 'Processing C: data demo.sft Vel0 = 4.0E+008 Vel = 4.0E+008'.

As the data is being processed, integer numbers starting from 0 increasing to 399 are written to the screen. This represents the 400 depth levels of the data being processed (data trace length = 400).

When the processing is completed the reconstructed image is plotted. This image is similar to the one shown in Figure 3-10. The image in Figure 3-10 shows two targets: The target on the left is at station 0+13 ft. with a depth of approximately 6 ft. and the target on the right is at station 0+8 ft. at a depth of 3 ft. The extended image between 11 to 12 ft. deep are noise or reflection from some bottom feature which has been magnified by the radar's time variable gain.

Note: The data is written into the disk as C: data demo.img.

Step 5 Depress 'Enter' to return to the T-SAFT menu.

Step 6 Depress 'ESC' to return to the function selection menu.

The above illustrates the processing of a sample radar data. Automatic processing and variations in data parameters can be easily accommodated with minor selection changes and parameter modifications.

3.12.7 Velocity Determination Demonstration

Step 1 From the function selection menu select the velocity program.

Observation: The velocity menu as in Figure 3-11 is displayed. Since we want to use C: data Demo.prv for this demonstration, no entry modification is needed.

Step 2 Depress 'F1' to execute the velocity determination routine.

Observation: The plot of the radar data generated by Preview is displayed. The right half of an upside down V is also shown at the left edge of the plot. The apex is used to mark the left and right terminal of the plot for the width calibration.

Step 3 Move the peak of the 'V' to a left known marker and depress 'F1'. In this case leave it at the left edge and depress 'F1'.

Step 4 Move the peak of the 'V' to a right known mark and depress 'F2'. In this case move it to the right edge of the plot and depress 'F2'.

Observation: The prompt reads 'enter distance between marks'.

Step 5 Enter '30' for 30 ft.

Observation: A second prompt reads 'enter radar sampling period (ns)'. The sampling period for the data is .16 ns.

Step 6 Enter '.16'.

Step 7 Press 'F3' to complete calibration. Note: Prior to depressing 'F3' if an error in the procedure is discovered, steps 3 to 6 can be re-executed.

Observation: The upside down 'V' begins blinking.

Step 8 Move the upside down 'V' (which is actually a hyperbola generated at depth = 0 ft.) to the first right target $x \simeq 23$.

Step 9 Move the template hyperbola down using the down arrow and observe the depth offset change. Stop the downward translation when depth offset = 23. The apex is now at the ground surface. The value was previously determined during field survey and data digitization.

Step 10 Using the page down key, reconstruct the template at the depth of the target. Observe z0; this gives the depth of the hyperbola.A fairly good match is obtained when the z0 $\simeq$ 2.6 ft.

Step 11 Move the template hyperbola to straddle the target near the middle of the display using the left-right arrows and the page up and down keys. A fairly good match is achieved when x = 17.38 and z0 = 6.144.

Step 12 Press 'F9' to write the velocity and depth offset back into the parameter header file.

Observation: The template hyperbola freezes and the prompt reads 'enter file to update'.

Step 13 Type file name and enter C: data Demo.hdr.

Observation: The template hyperbola begins to blink again and the prompt changes to 'Press 'F9' to update file parameters, 'F10' to quit'.

Discussion: The default value for the velocity of GPR propagation is set at 0.400 ft./ns and the matching is fairly well because the defaults have been selected using this data. If the default velocity was 0.35 ft./ns or 0.45 ft./ns then the matching would be not so perfect. To illustrate this let us change the velocity to .45 and try to match the hyperbola again.

Step 14 To increase the velocity depress the 'home' key. To decrease the velocity depress the 'end' key. Depress 'home' until Vel = 0.4500.

Observation: The template hyperbola is moved up. Since the velocity has increased 6.144 ft. is closer to the beginning of the data trace, therefore we have to move it back down by increasing z0.

Step 15 Depress or tap 'page down' until the apex of the template hyperbola sits on top of the target (z0 = 6.792). Notice that the wings of the template no longer hugs the wings of the target hyperbola. A remedy for this is to generate the template at a shallower depth and translating the template with the up/down arrow keys.

Step 16 Move the depth up to 5.424 ft. then translate the template down to depth offset = 63. These numbers given have been obtained by iterating between the depth and offset controls.

Observation: The template and target matches fairly well. Though each template with its set of parameters is unique the ideal target hyperbola shape and width permits more than one template match. This uncertainty can be remedied by crosschecking with the field recorded depth offset and a second target of different depth.

Step 17 Move template to x = 22.84 and zo = 1.536.

Observation: In this position the template peak coincides with the template peak. However, the wings of the hyperbola are clipping the wings of the target hyperbola. (A little visualization may be helpful.)

Discussion: In this situation we know the velocity is too high, however, without apriori knowledge the template can be matched to both targets by iterating between the two targets until the best compromise can be found. With further practice one can estimate the velocity and depth offset fairly well. However, there is not substitute for careful field measurement of depth offset and velocity.

Step 18 Return to Function selection menu by depressing 'F10' then 'ESC'.

3.12.8 Directory Execution

Step 1 Select directory function using arrow key.

Step 2 Press 'F1' to execute.

Observation: A 'file mask:' prompt is displayed.

Step 3 - List the current directory by depressing 'Enter'.

Observation: A display similar to Figure 3-13 is shown on the screen.

Step 4 - Depress 'Enter' to return to function selection menu.

3.12.9 Review Execution

Step 1 Select review function from Function selection menu.

Step 2 Press 'F1' to execute.

Observation: The review menu as shown in Figure 3-14 is displayed.

Step 3 Modify the data file name if desired. In this example C: data demo is the correct name to use. Select any of the options, for example, the reconstructed image option using the arrow keys.

Step 4 Press 'F1' to display the reconstructed image previously generated.

Observation: The T-SAFT reconstructed image similar to Figure 3-10 is displayed. A prompt asks if a print of the display is desired.

Step 5 Return to the review menu by depressing 'Enter'.

Observation: Review menu is displayed.

Step 6 Return to function selection menu by depressing 'ESC'.

Discussion: Parameter values can be changed when option 4 and 5 are selected in Review. To change parameter, answer 'Y' modify value prompt.

3.12.10 Autoproc Execution

Step 1 Select autoproc function using arrow keys.

Step 2 Depress 'F1' to execute.

Step 3 Enter data file name, output file name, and all parameters. Select processing and print option. For this example, the parameters and selections are correct with the exception that the depth offset should be 23.

Step 4 Press 'F1' to start execution.

Observation: Preview plots the radar survey data line by line then stops for the width calibrations since the option 'Y' for calibration is selected.

Step 5 Calibrate width as in the Preview example. At the conclusion of the calibration the following displays will be shown.

Observations:

1. Preview radar data screen is redisplayed with the horizontal data cursor at left.

2. A detailed plot of this spanned data is made on a subsequent screen.

3. The Preview radar data is redisplayed with the horizontal data cursor somewhere in the middle.

4. A detailed plot of this spanned data is made.

5. The Preview radar data is redisplayed with the horizontal data cursor at the right third of the data.

6. T-SAFT reads first data set. C: data demo0.sft. T-SAFT displays processing index 0 to 399.

7. T-SAFT plots first reconstructed image.

* Steps 6 and 7 are repeated for next two data sets Demo1.sft and Demo2.sft.

For this example there are only three data sets. A longer survey may produce more data sets. In general, the software segments the data into 10 ft. per reconstruction.

When the data is exhausted the program ends in the T-SAFT menu.

8. T-SAFT menu is redisplayed.

Note: All the intermediate and data files are recorded on disk under C: data demo?.???. Printing would have been carried out if the option was selected.

Step 6 Press 'ESC' to return to function selection menu.

Step 7 Press 'ESC' to go back to the NCEL title screen.

Step 8 Press 'ESC' to return control to DOS.

This completes the step by step example data processing of a real GPR survey.

3.13 T-SAFT RECONSTRUCTED IMAGE INTERPRETATION

The interpretation of the reconstructed image is simple after some familiarity is gained through practice with known data. A good training set is the Battelle Northwest surveyed data included here as Demo.dat. A pictorial description of the target is shown in Figure 3-16. The radar survey data is shown in Figure 3-5. The subsequent T-SAFT processing yields the image shown in Figure 3-10.

3.13.1 Image Annotations

In Figure 3-10, the left vertical scale is the depth scale in feet. The horizontal scale represents the survey footage marking. The left most number after 'sta' is the hundred feet number. The parameters on the right are the file name of the data, the time and date of the reconstruction, the width represented by the reconstruction, the depth represented by the data, the depth offset from the surface of the ground, the velocity of propagation of the radar and the digitization sample period. Imax and imin are maximum and minimum intensity values of the reconstructed image. These numbers are used as a guide in comparing closely similar reconstructions. In general, the higher the maximum intensity the more focussed the data. X and Y display the current pixel being plotted and would always show 31 and 180 when the plot is completed.

3.13.2 The Image

The image within this 20 ft. deep by 10 ft. wide border will scale itself according to actual survey dimensions. In Figure 3-10, the left image at approximately sta0+13 is slightly over 6 ft. deep.

3.13.2.1 Target Recognition

A pipe or other elongated target traversed perpendicularly is characterized by multiple cycles of hyperbolic pattern as shown in the 2D plot of the data (see Figure 3-5). The hyperbolic pattern is focussed by T-SAFT into the multiband characteristic of a pipe target. The reconstruction in Figure 3-10 shows two such multiband targets at 6 and 3 ft. depth.

A long extended image as shown between the depth of 11 and 12 ft. is caused by a combination of an increased gain in the radar and the undergound features. It is useful to refer to the available data depth (e.g., 12.8 ft.) to make this determination. In general, targets imaged close to the bottom of available data depth should be interpreted with caution.

3.13.2.2 Target Sizing

This target is weaker than the one on the right. This target is fairly well focussed in spite of the dissected hyperbolic input target signal shown in Figure 3-8. This is possible due to T-SAFT's capability to refocus the peripheral signals. This target under discussion is a weak plastic target. The size of the target is 4.5 inches in diameter according to the layout drawing of Figure 3-16. Using the 300 MHz, GSSI antenna the resultant wavelength is approximately two feet. Theoretically, this wave length can only provide target resolution down to one foot ($\frac{\lambda}{2}$) under ideal conditions. Thus, the resultant image has, at best, a one foot resolution. The target images reconstructed are exceedingly sharp under this constraint.

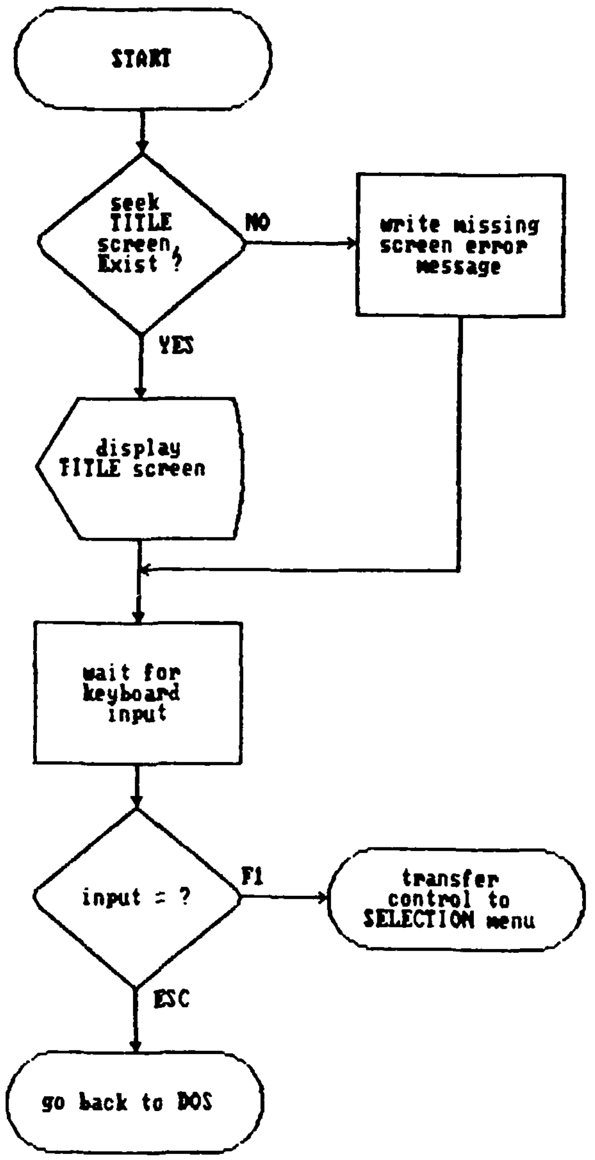
The target image on the right at sta+8 is at approximately 3 ft. deep. The layout shows a 1.9 inch steel pipe at 2.7 inches deep. Though one can infer a smaller target by the more compact horizontal extend of the reconstructed image, it is testing the theoretical resolution limits. Better size resolution can be obtained by using a higher frequency radar. The tradeoff is the depth of penetration.

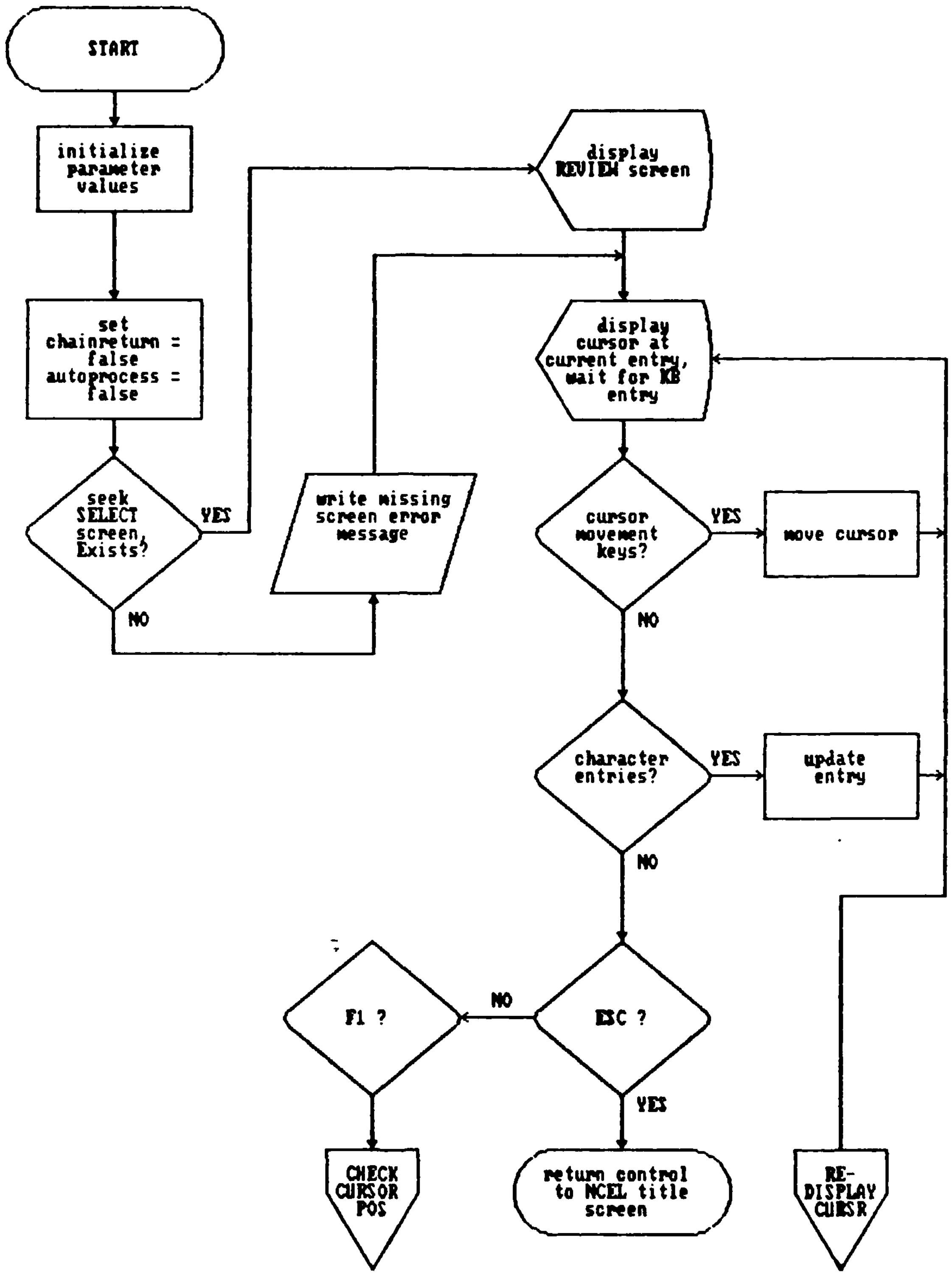
3.13.2.3 Target Composition

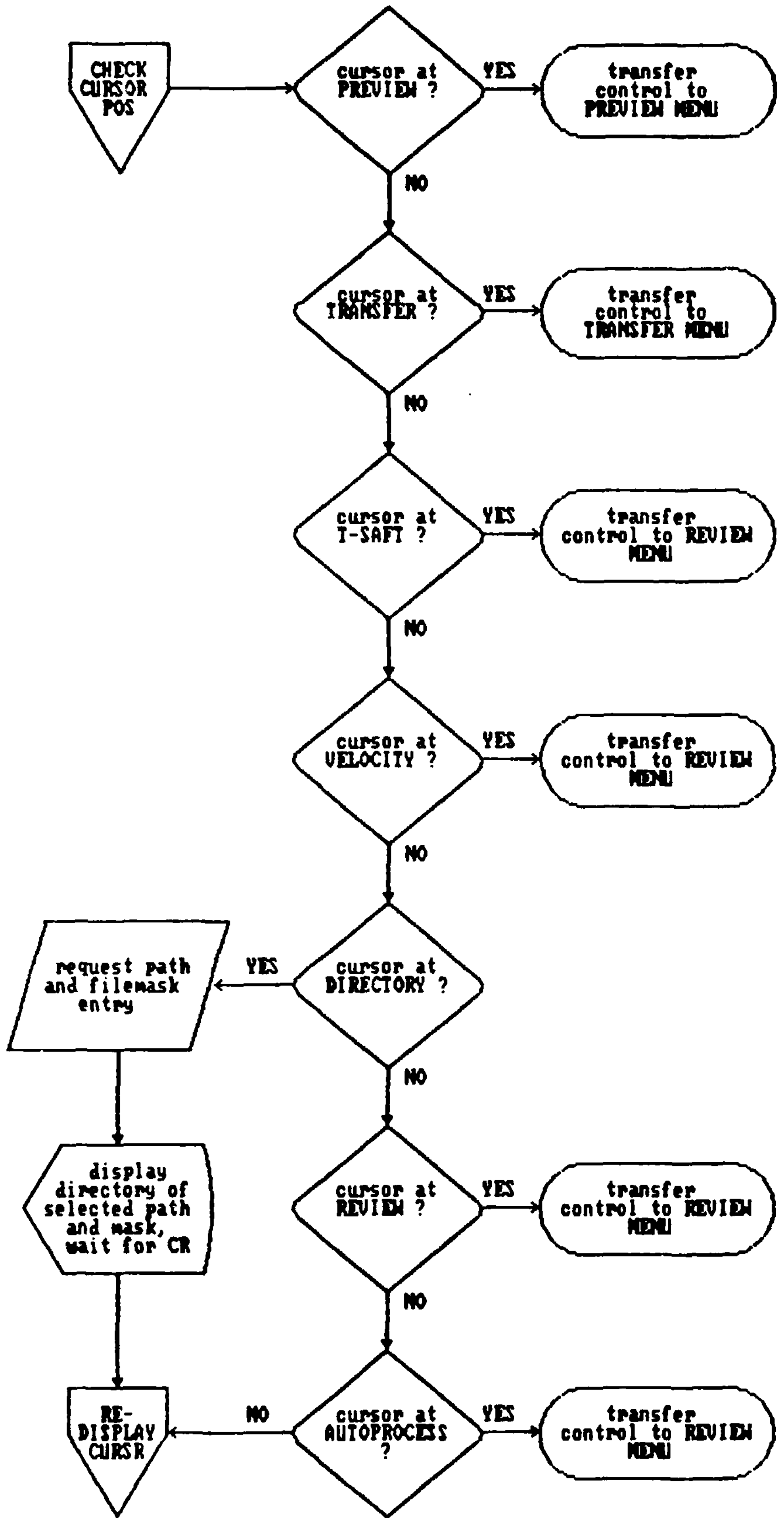
In theory, the target material can determine the polarity of the target signal. In practice, however, it has been observed that the polarity of the returned radar signal is influenced by the complex nonuniform transfer function of the soil and is not a wholly reliable indicator of target.

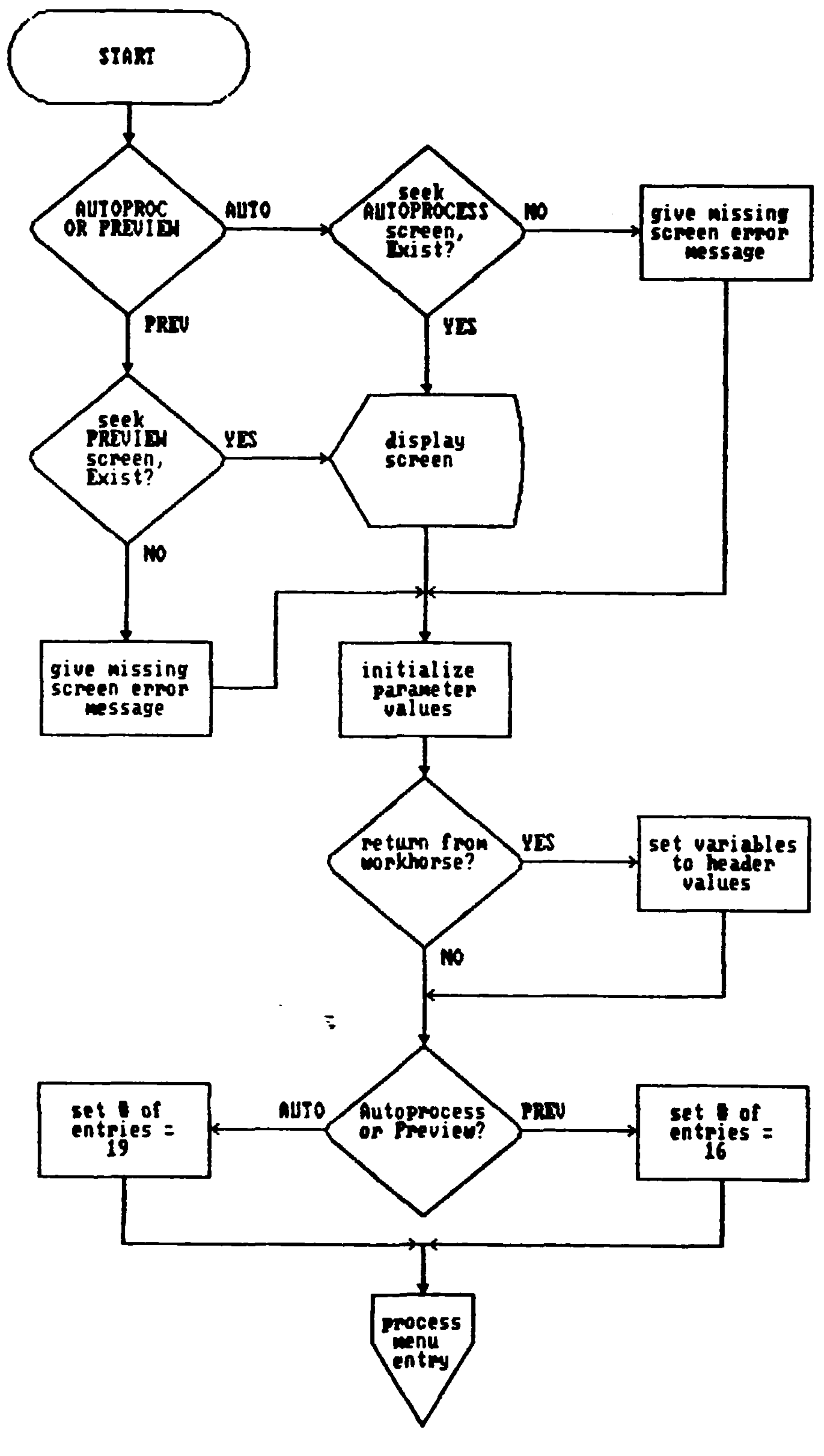
4.0 NCEL RADAR IMAGING SOFTWARE FLOW CHARTS

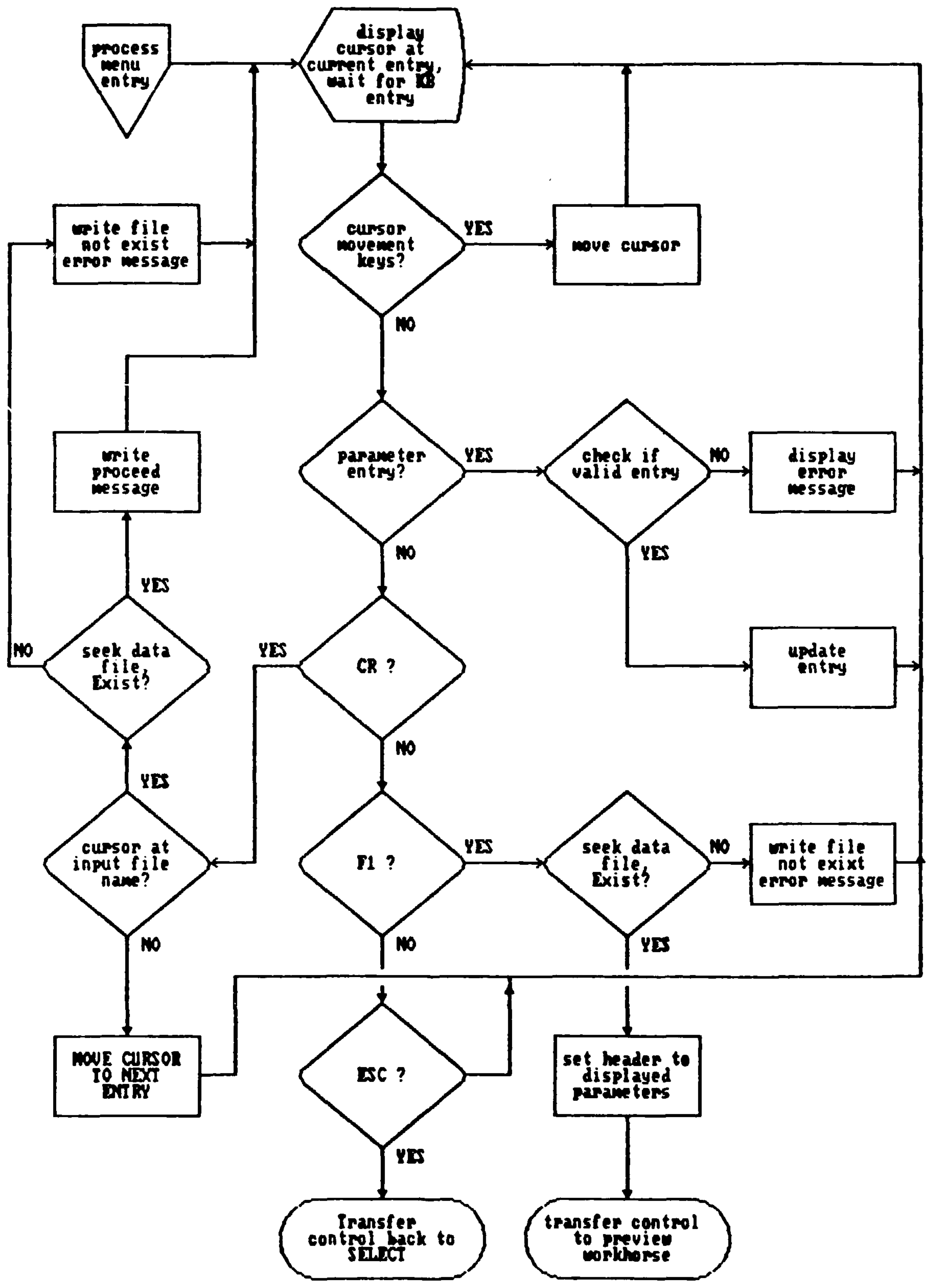
NCEL TITLE MENU PROGRAM

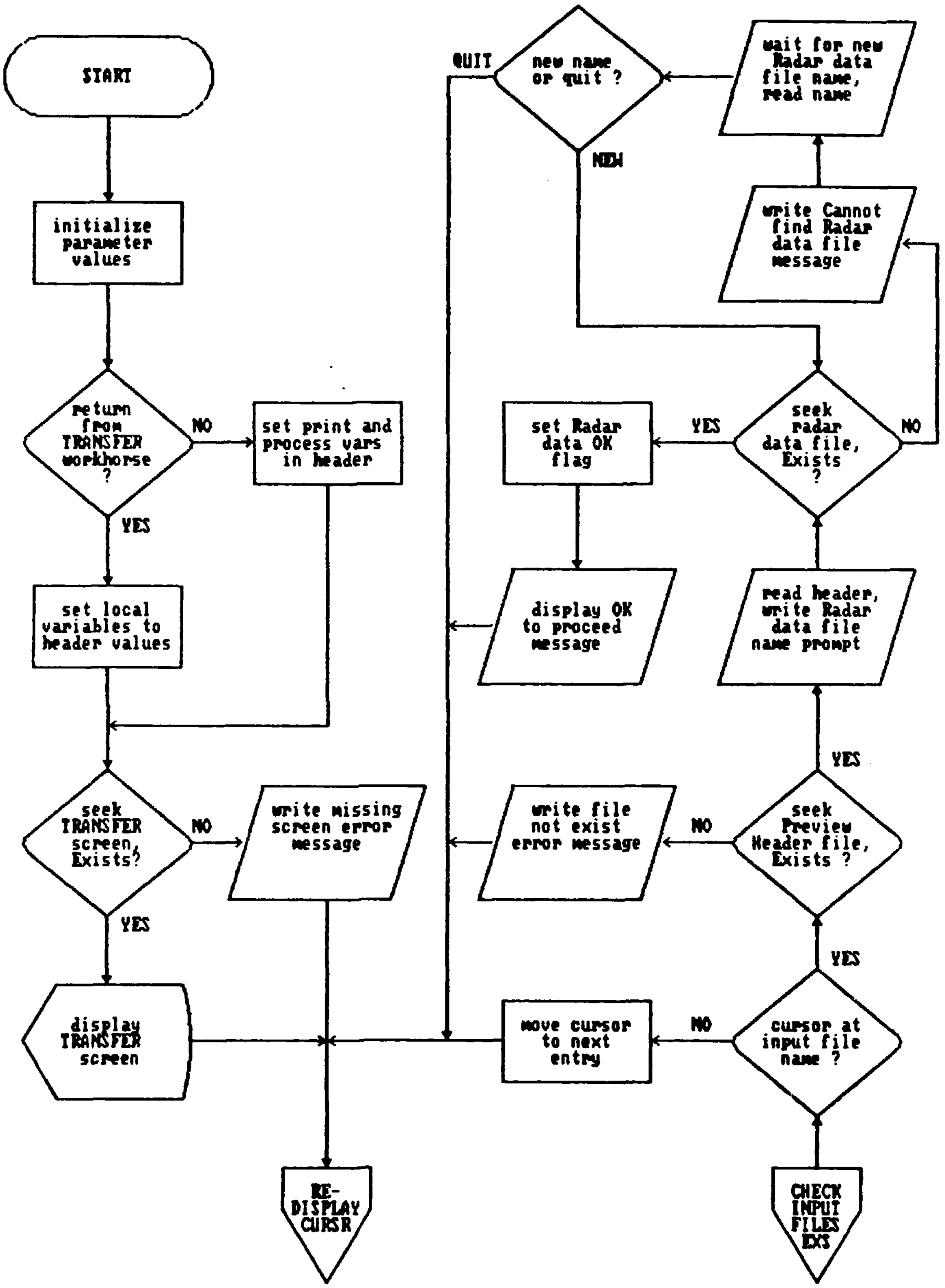

START
seek TITLE screen, Exist ?
NO
write missing screen error message
YES
display TITLE screen
wait for keyboard input
input = ?
F1
transfer control to SELECTION menu
ESC
go back to DOS

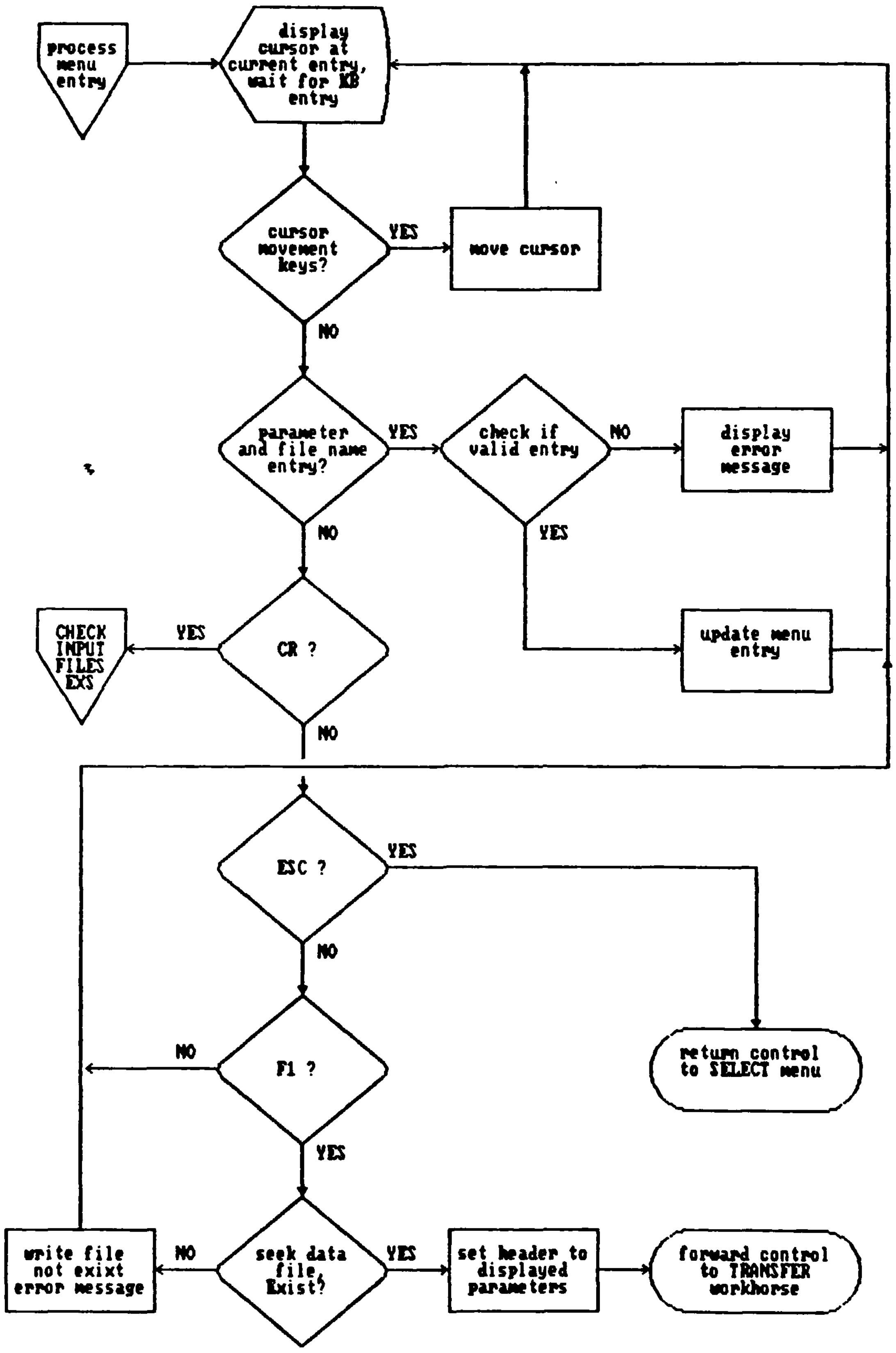

START
initialize parameter values
set chainreturn = false autoprocess = false
seek SELECT screen, Exists?
YES
NO
write missing screen error message
display REVIEW screen
display cursor at current entry, wait for KB entry
cursor movement keys?
YES
move cursor
NO
character entries?
YES
update entry
NO
ESC ?
NO
F1 ?
YES
CHECK CURSOR POS
return control to NCEL title screen
RE-DISPLAY CURSR

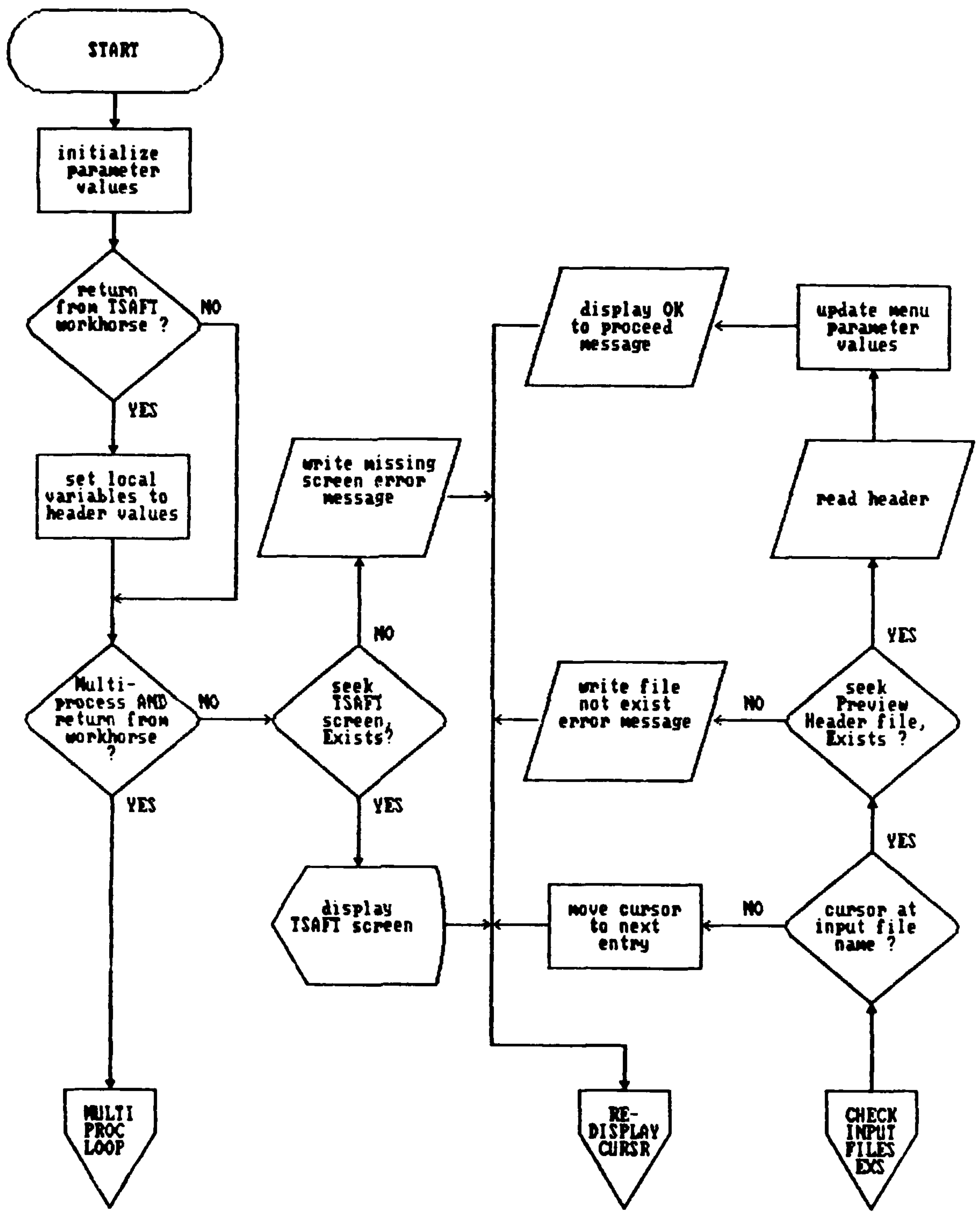

CHECK CURSOR POS
cursor at PREVIEW ?
YES
transfer control to PREVIEW MENU
NO
cursor at TRANSFER ?
YES
transfer control to TRANSFER MENU
NO
cursor at T-SAFT ?
YES
transfer control to REVIEW MENU
NO
cursor at VELOCITY ?
YES
transfer control to REVIEW MENU
NO
cursor at DIRECTORY ?
YES
request path and filemask entry
display directory of selected path and mask, wait for CR
NO
cursor at REVIEW ?
YES
transfer control to REVIEW MENU
NO
cursor at AUTOPROCESS ?
YES
transfer control to REVIEW MENU
NO
RE-DISPLAY CURSR

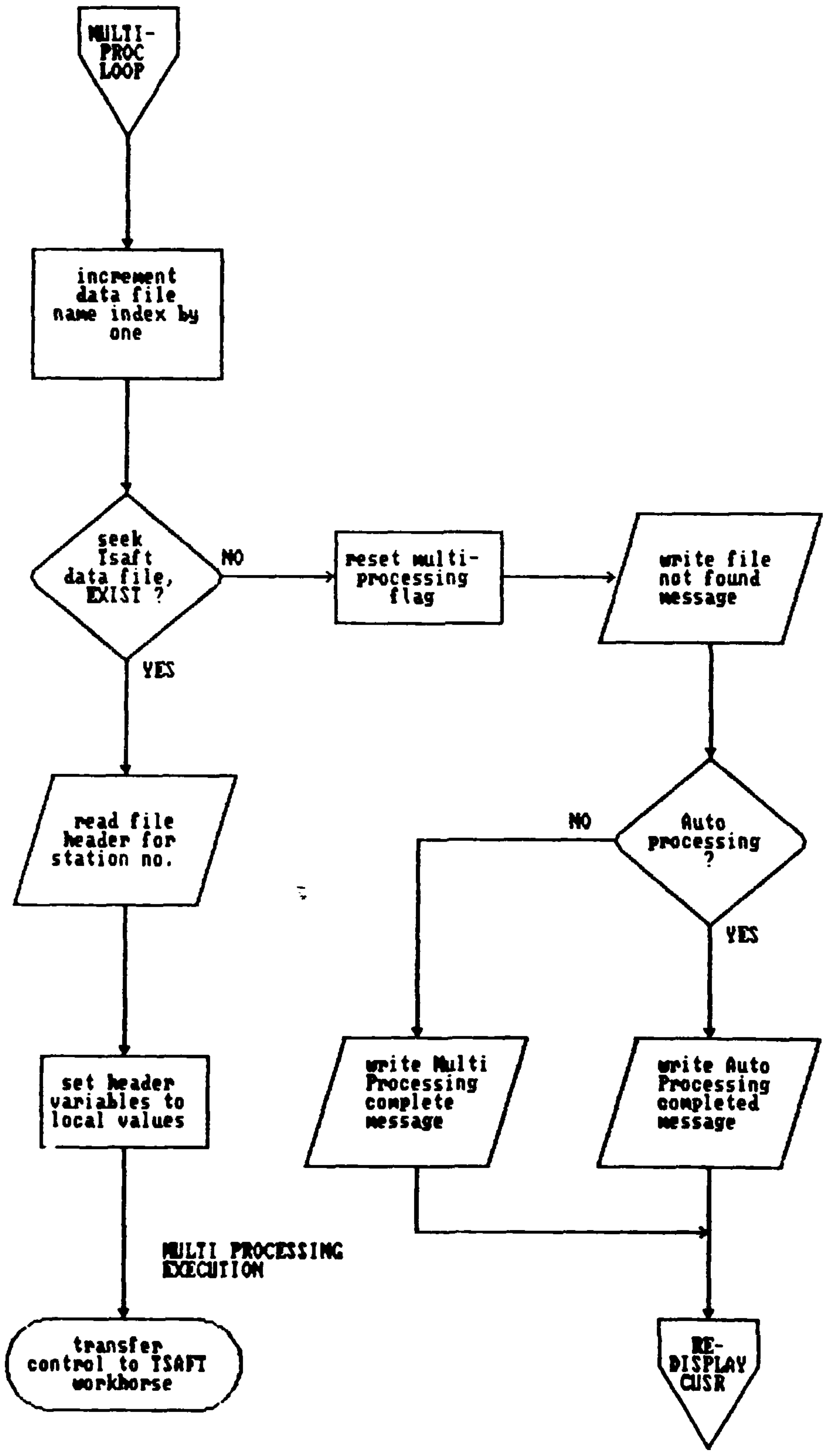

START
AUTOPROC OR PREVIEW
AUTO
seek AUTOPROCESS screen, Exist?
NO
give missing screen error message
YES
PREV
seek PREVIEW screen, Exist?
YES
display screen
NO
give missing screen error message
initialize parameter values
return from workhorse?
YES
set variables to header values
NO
Autoprocess or Preview?
AUTO
set # of entries = 19
PREV
set # of entries = 16
process menu entry

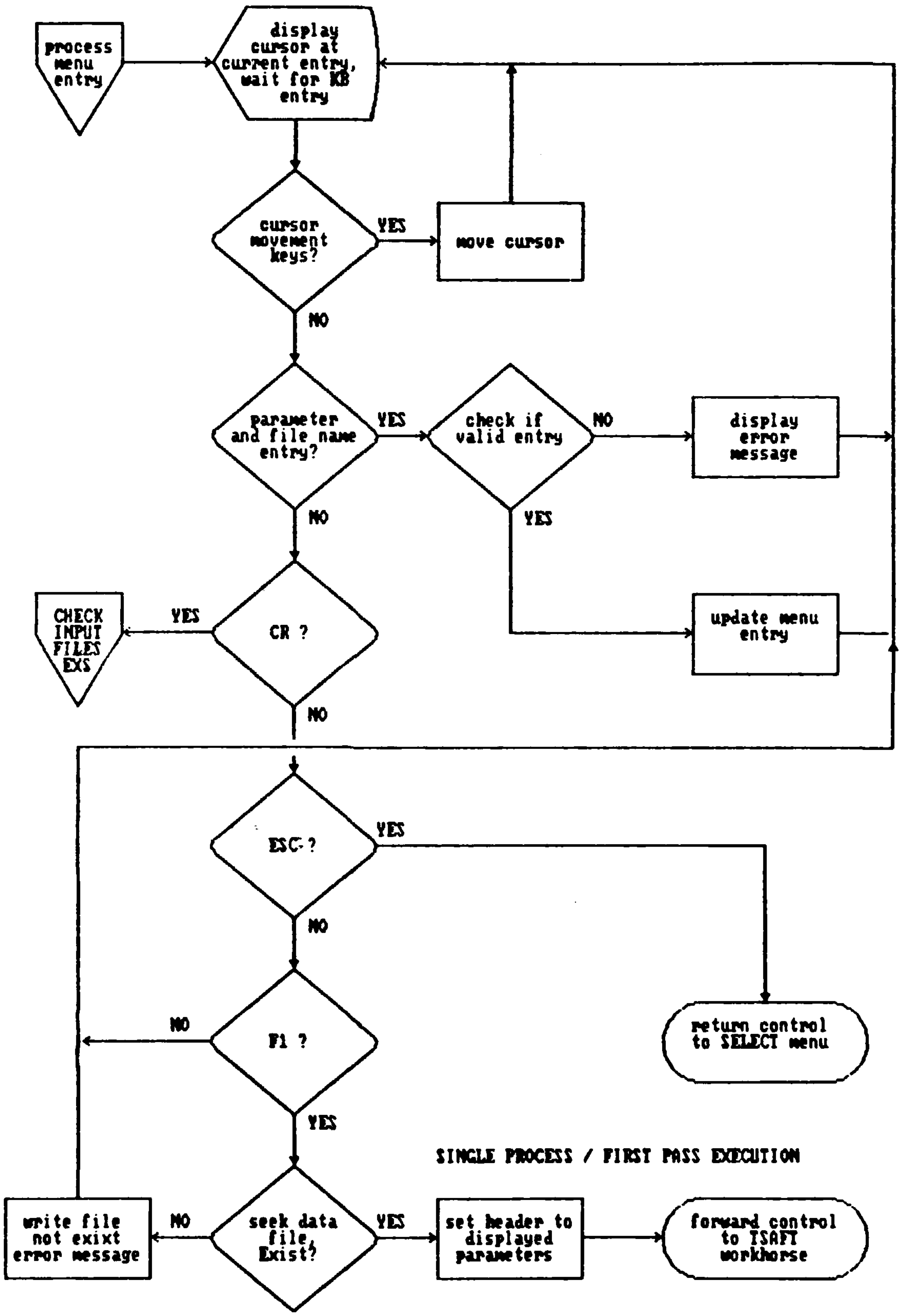

process menu entry
display cursor at current entry, wait for KB entry
cursor movement keys?
YES
move cursor
NO
parameter entry?
YES
check if valid entry
NO
display error message
YES
update entry
NO
CR ?
YES
NO
F1 ?
YES
seek data file, Exist?
NO
write file not exixt error message
YES
set header to displayed parameters
transfer control to preview workhorse
NO
ESC ?
YES
Transfer control back to SELECT
cursor at input file name?
YES
NO
MOVE CURSOR TO NEXT ENTRY
seek data file, Exist?
YES
NO
write proceed message
write file not exist error message

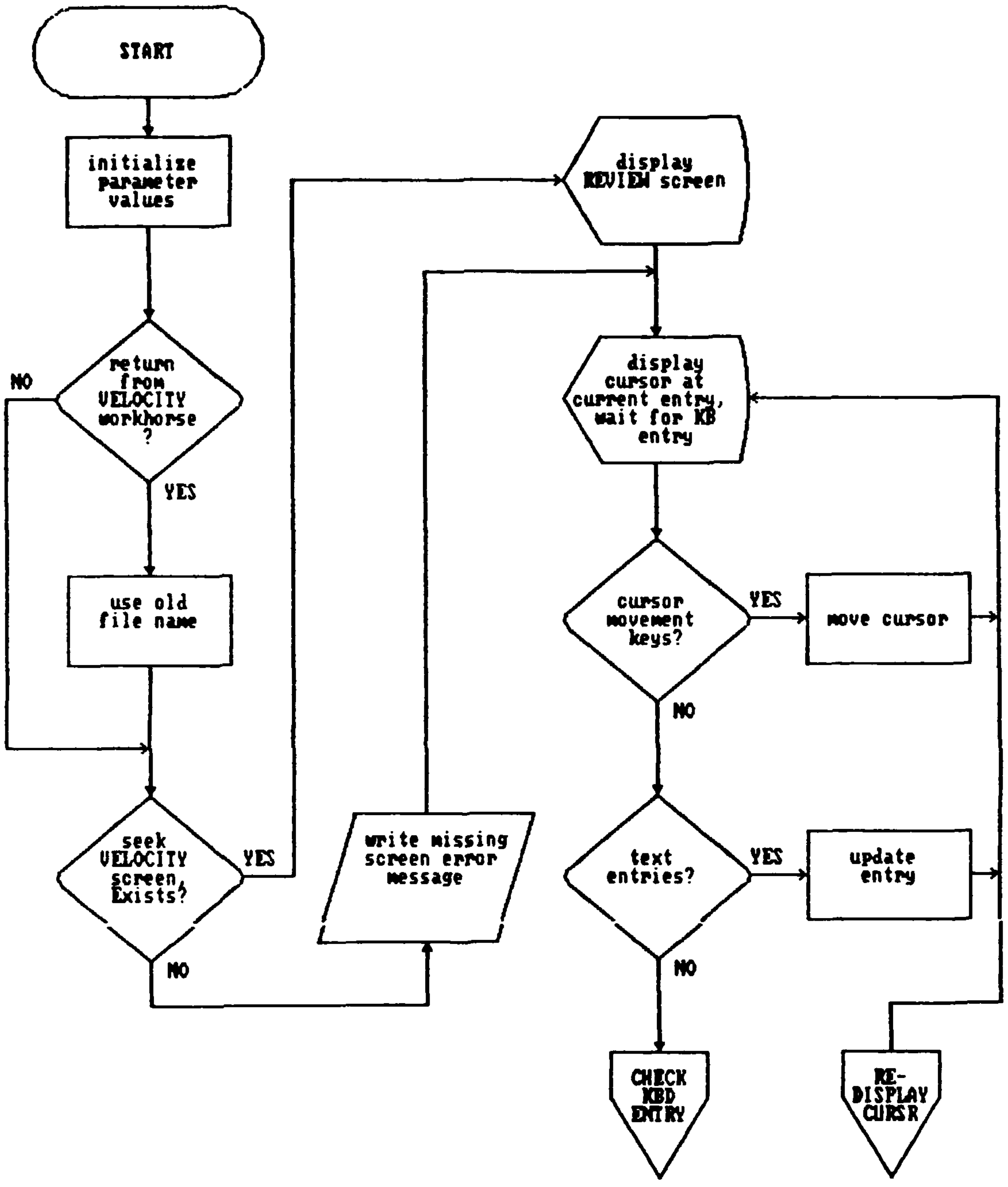

START
initialize parameter values
return from TRANSFER workhorse ?
NO
YES
set print and process vars in header
set local variables to header values
seek TRANSFER screen, Exists?
NO
YES
write missing screen error message
display TRANSFER screen
RE-DISPLAY CURSR
QUIT
new name or quit ?
NEW
wait for new Radar data file name, read name
write Cannot find Radar data file message
seek radar data file, Exists ?
YES
NO
set Radar data OK flag
display OK to proceed message
read header, write Radar data file name prompt
YES
seek Preview Header file, Exists ?
NO
write file not exist error message
YES
cursor at input file name ?
NO
move cursor to next entry
CHECK INPUT FILES EXS

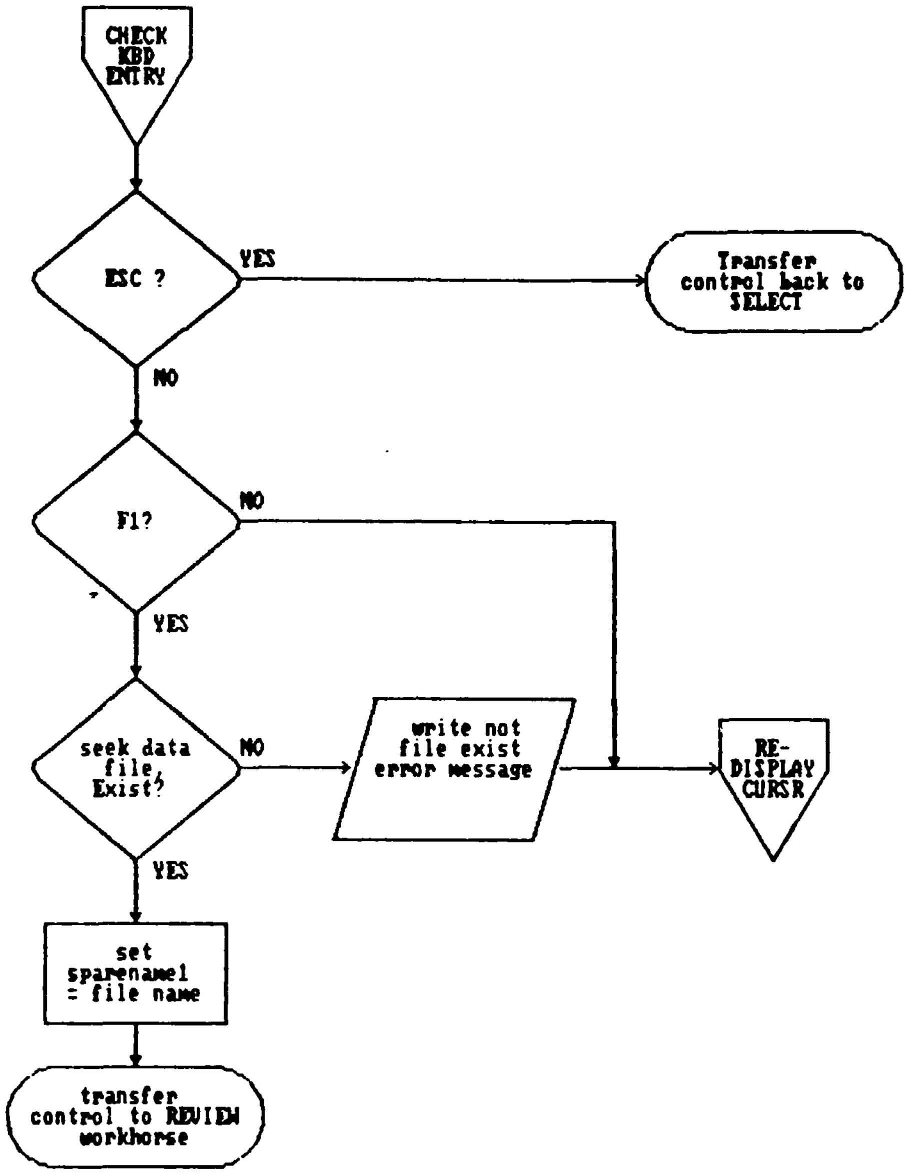

process menu entry
display cursor at current entry, wait for KB entry
cursor movement keys?
YES
move cursor
NO
parameter and file name entry?
YES
check if valid entry
NO
display error message
YES
update menu entry
NO
CR ?
YES
CHECK INPUT FILES EXS
NO
ESC ?
YES
return control to SELECT menu
NO
F1 ?
NO
YES
seek data file, Exist?
NO
write file not exixt error message
YES
set header to displayed parameters
forward control to TRANSFER workhorse

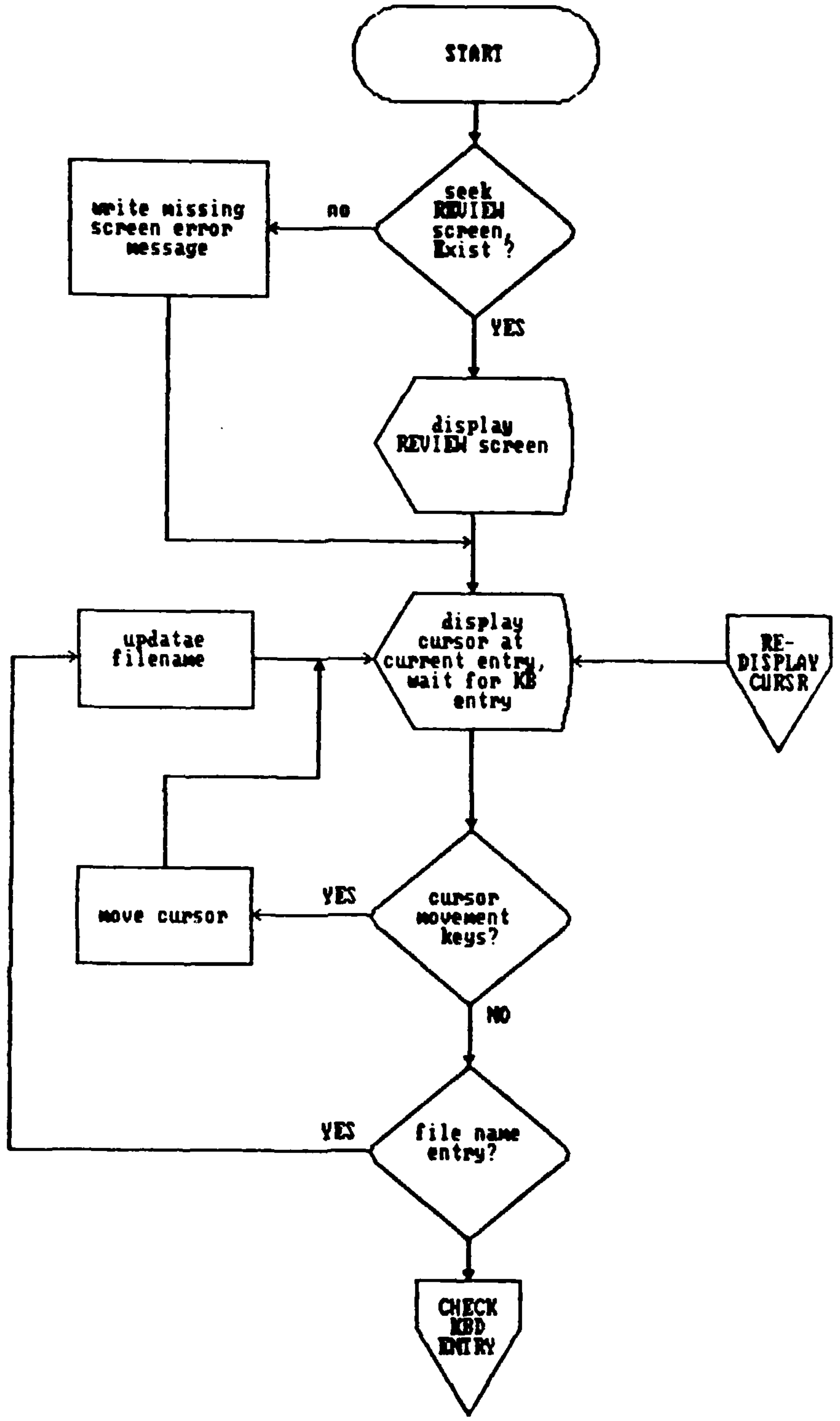

START
initialize parameter values
return from TSAFT workhorse ?
NO
YES
set local variables to header values
Multi-process AND return from workhorse ?
NO
YES
MULTI PROC LOOP
write missing screen error message
NO
seek TSAFT screen, Exists?
YES
display TSAFT screen
display OK to proceed message
update menu parameter values
read header
YES
write file not exist error message
NO
seek Preview Header file, Exists ?
YES
move cursor to next entry
NO
cursor at input file name ?
RE-DISPLAY CURSR
CHECK INPUT FILES EXS

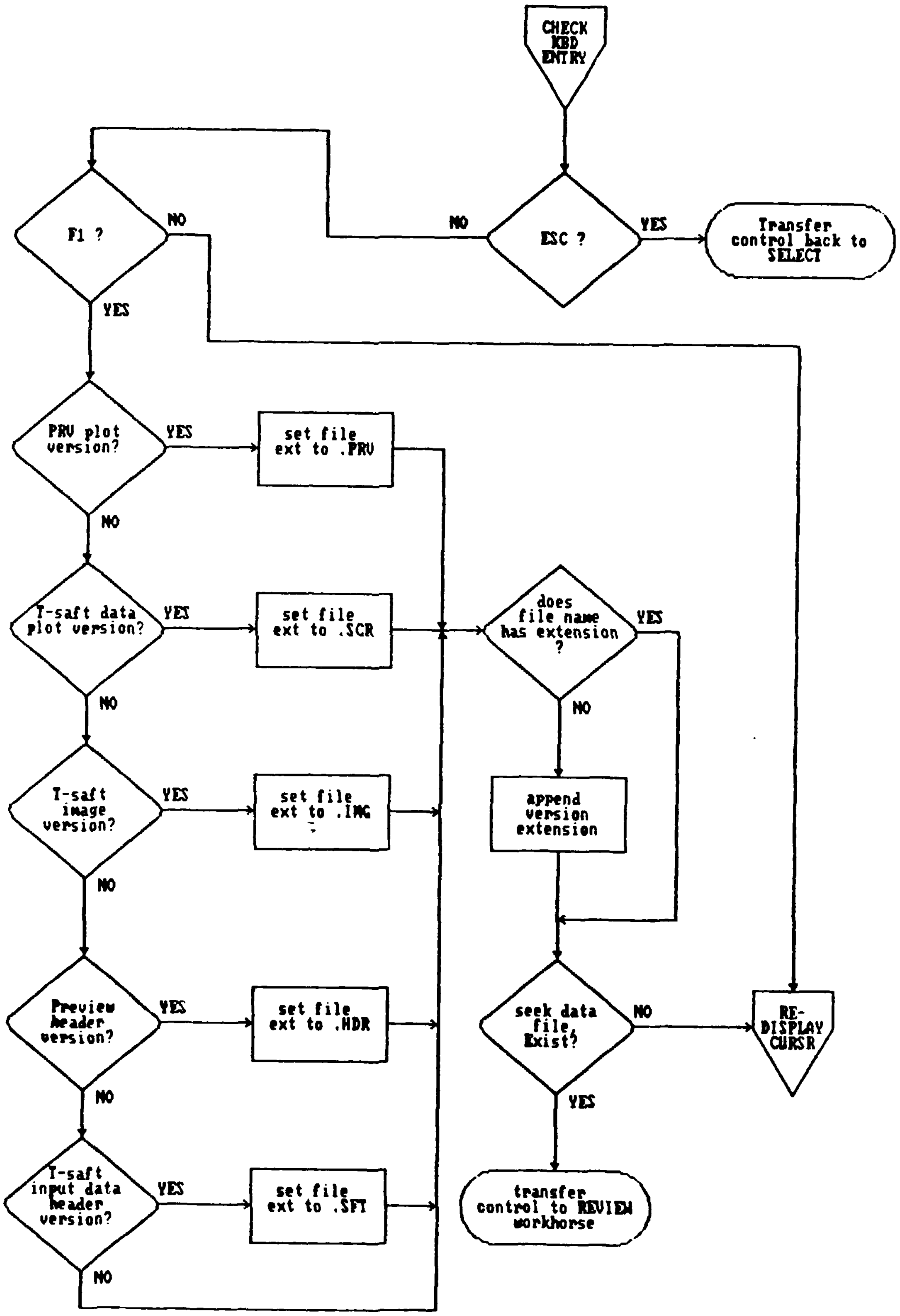

MULTI-PROC LOOP
increment data file name index by one
seek Tsaft data file, EXIST ?
NO
YES
reset multi-processing flag
write file not found message
Auto processing ?
NO
YES
read file header for station no.
set header variables to local values
write Multi Processing complete message
write Auto Processing completed message
MULTI PROCESSING EXECUTION
transfer control to TSAFT workhorse
RE-DISPLAY CUSR

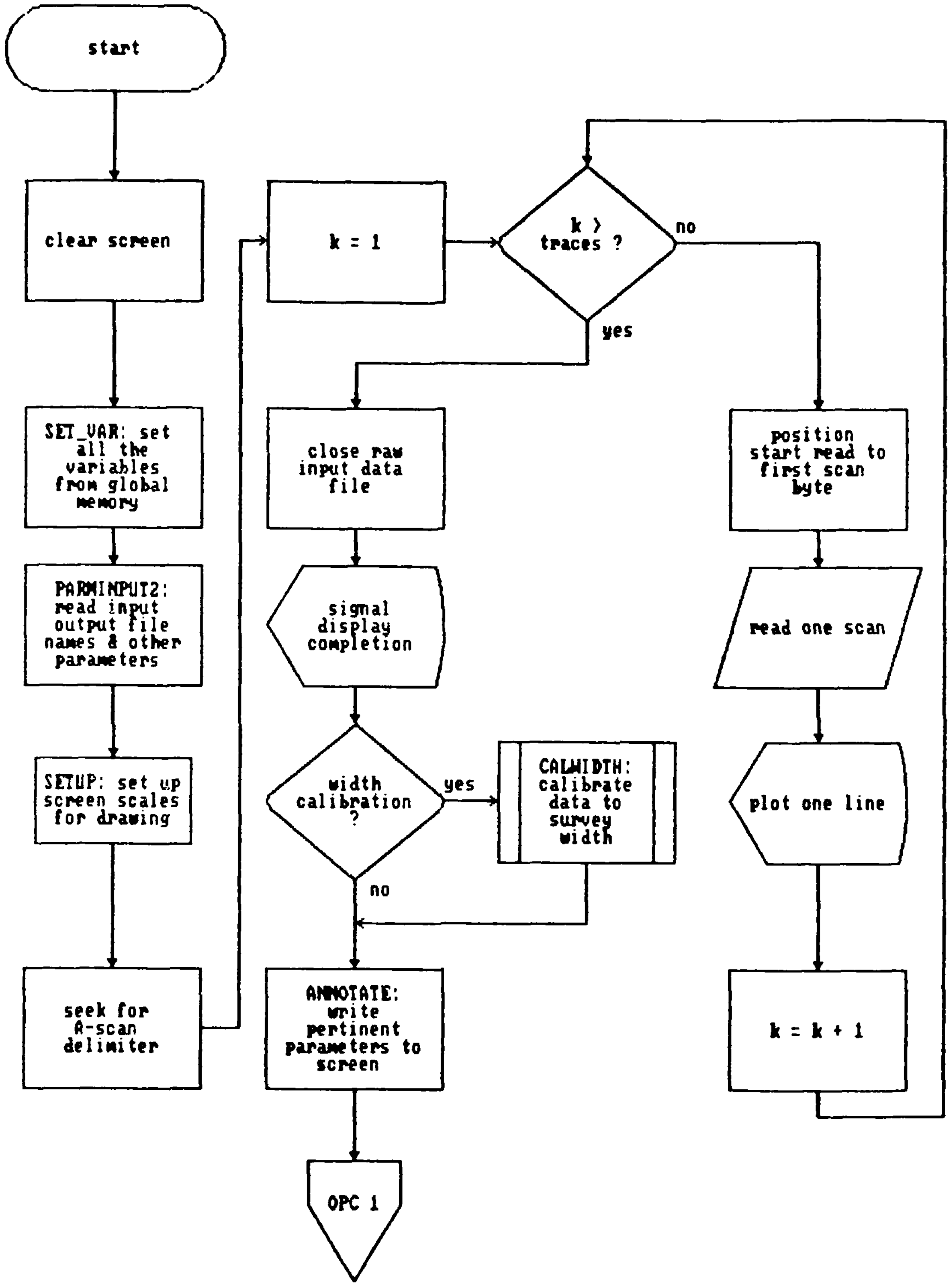

process menu entry
display cursor at current entry, wait for KB entry
cursor movement keys?
YES
move cursor
NO
parameter and file name entry?
YES
check if valid entry
NO
display error message
YES
update menu entry
NO
CR ?
YES
CHECK INPUT FILES EXS
NO
ESC ?
YES
return control to SELECT menu
NO
F1 ?
NO
YES
SINGLE PROCESS / FIRST PASS EXECUTION
seek data file, Exist?
NO
write file not exixt error message
YES
set header to displayed parameters
forward control to TSAFT workhorse

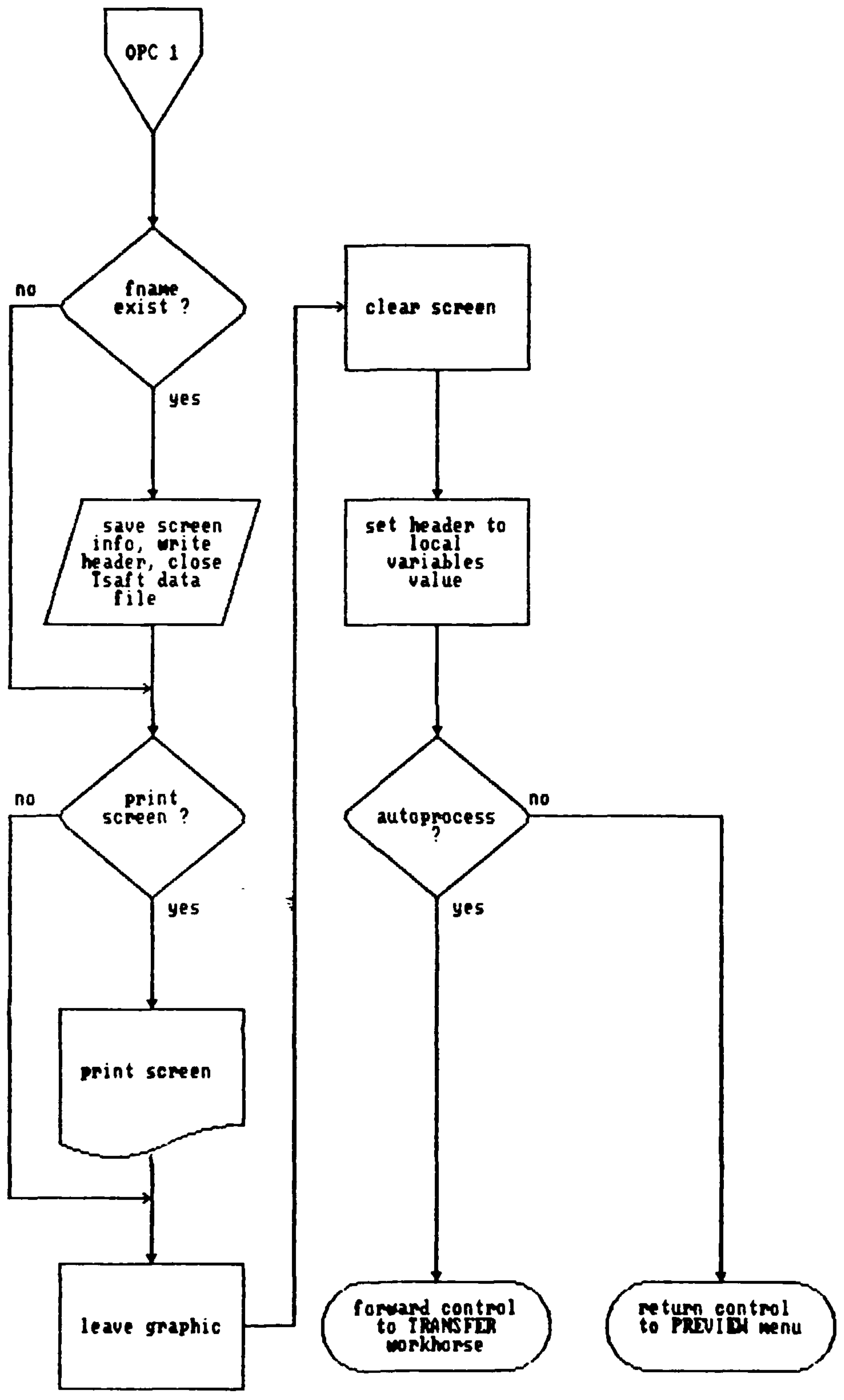

START
initialize parameter values
return from VELOCITY workhorse ?
NO
YES
use old file name
seek VELOCITY screen, Exists?
YES
NO
write missing screen error message
display REVIEW screen
display cursor at current entry, wait for KB entry
cursor movement keys?
YES
move cursor
NO
text entries?
YES
update entry
NO
CHECK KBD ENTRY
RE-DISPLAY CURSR

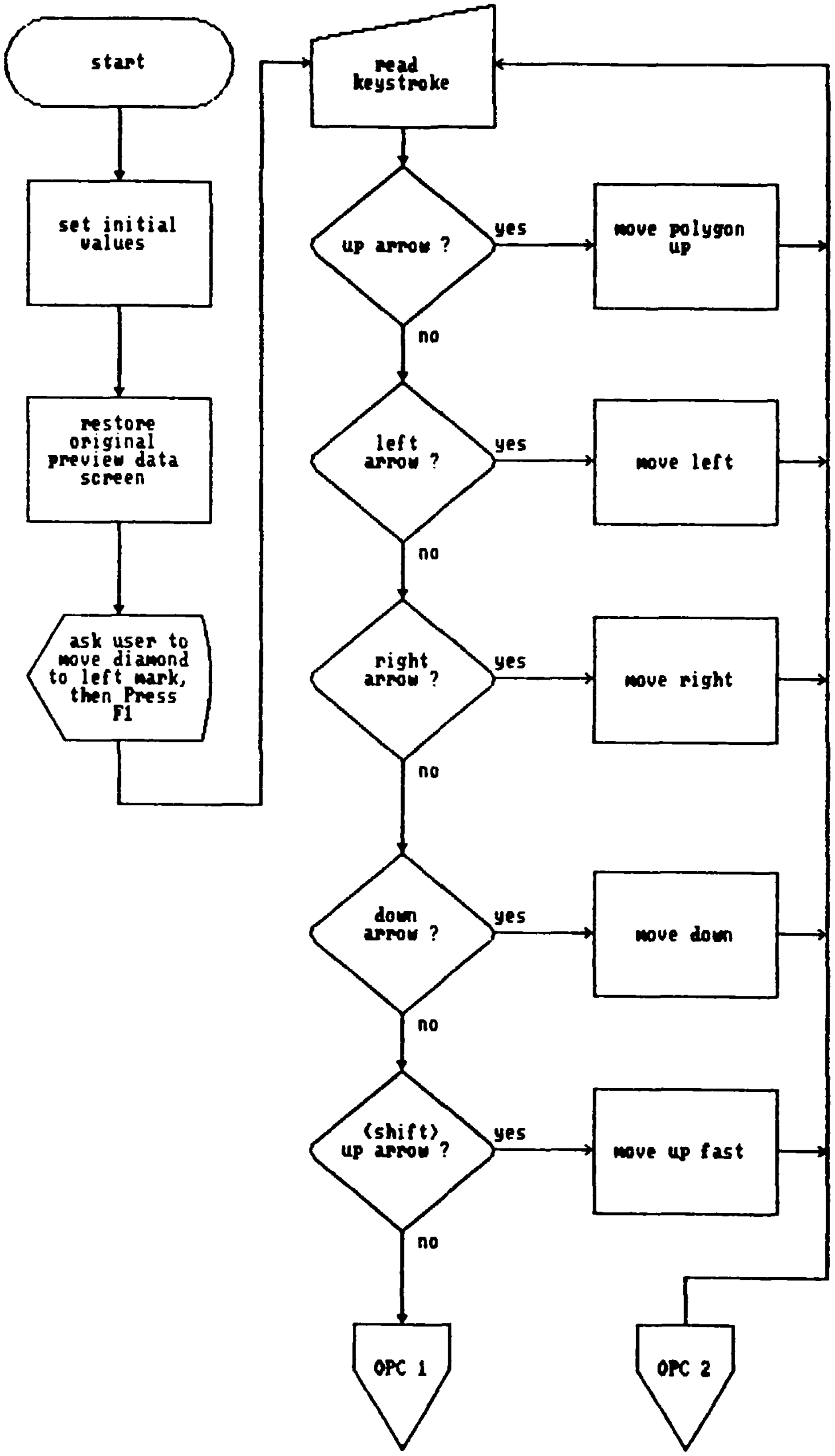

CHECK KBD ENTRY
ESC ?
YES
Transfer control back to SELECT
NO
F1?
NO
YES
seek data file, Exist?
NO
write not file exist error message
RE-DISPLAY CURSR
YES
set sparename1 = file name
transfer control to REVIEW workhorse

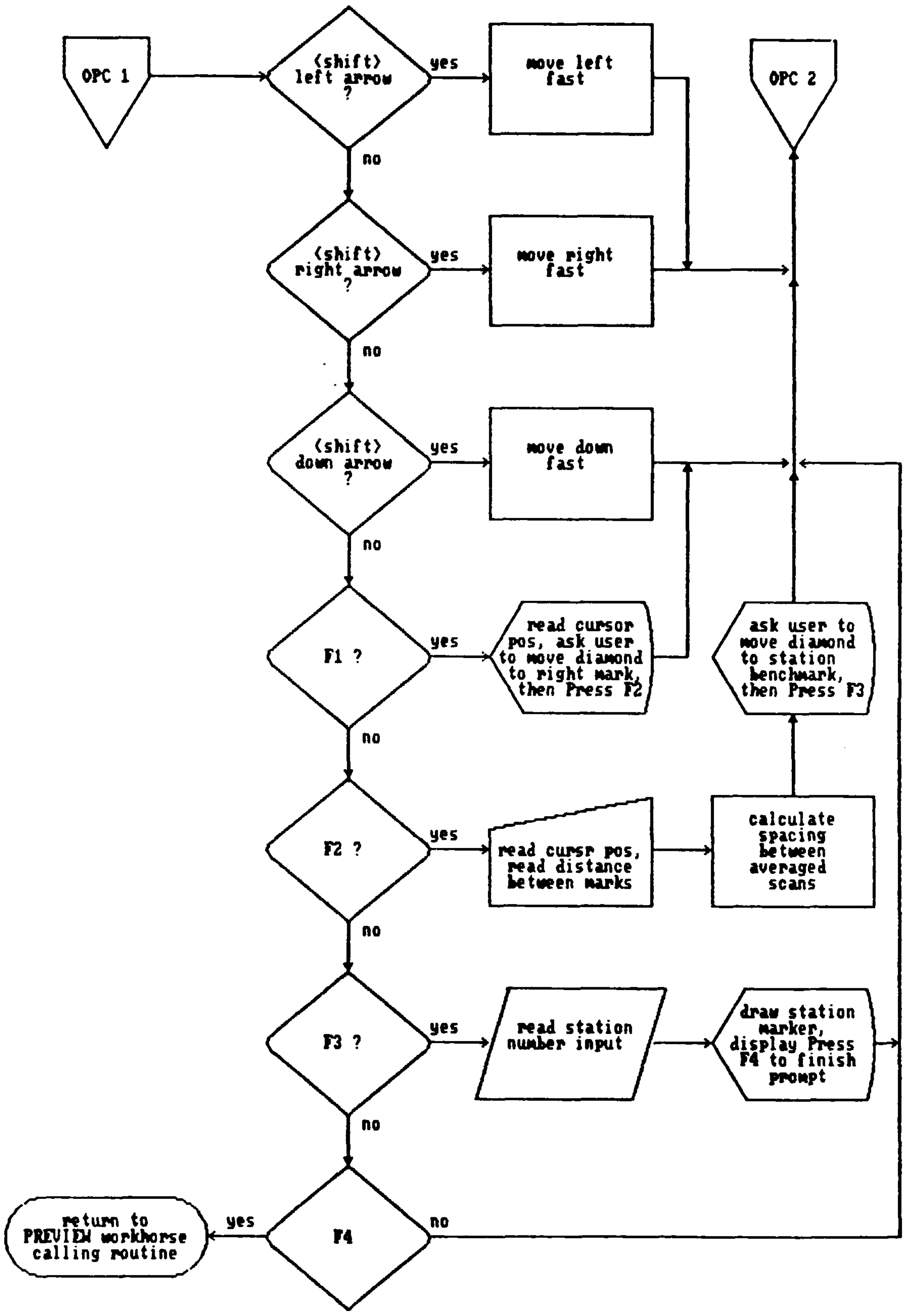

START
seek REVIEW screen, Exist ?
no
write missing screen error message
YES
display REVIEW screen
display cursor at current entry, wait for KB entry
updatae filename
RE-DISPLAY CURSR
cursor movement keys?
YES
move cursor
NO
file name entry?
YES
CHECK KBD ENTRY


CHECK KBD ENTRY
ESC ?
YES
Transfer control back to SELECT
NO
F1 ?
NO
YES
PRU plot version?
YES
set file ext to .PRU
NO
T-saft data plot version?
YES
set file ext to .SCR
NO
T-saft image version?
YES
set file ext to .IMG
NO
Preview header version?
YES
set file ext to .HDR
NO
T-saft input data header version?
YES
set file ext to .SFT
NO
does file name has extension ?
YES
NO
append version extension
seek data file, Exist?
NO
RE-DISPLAY CURSR
YES
transfer control to REVIEW workhorse

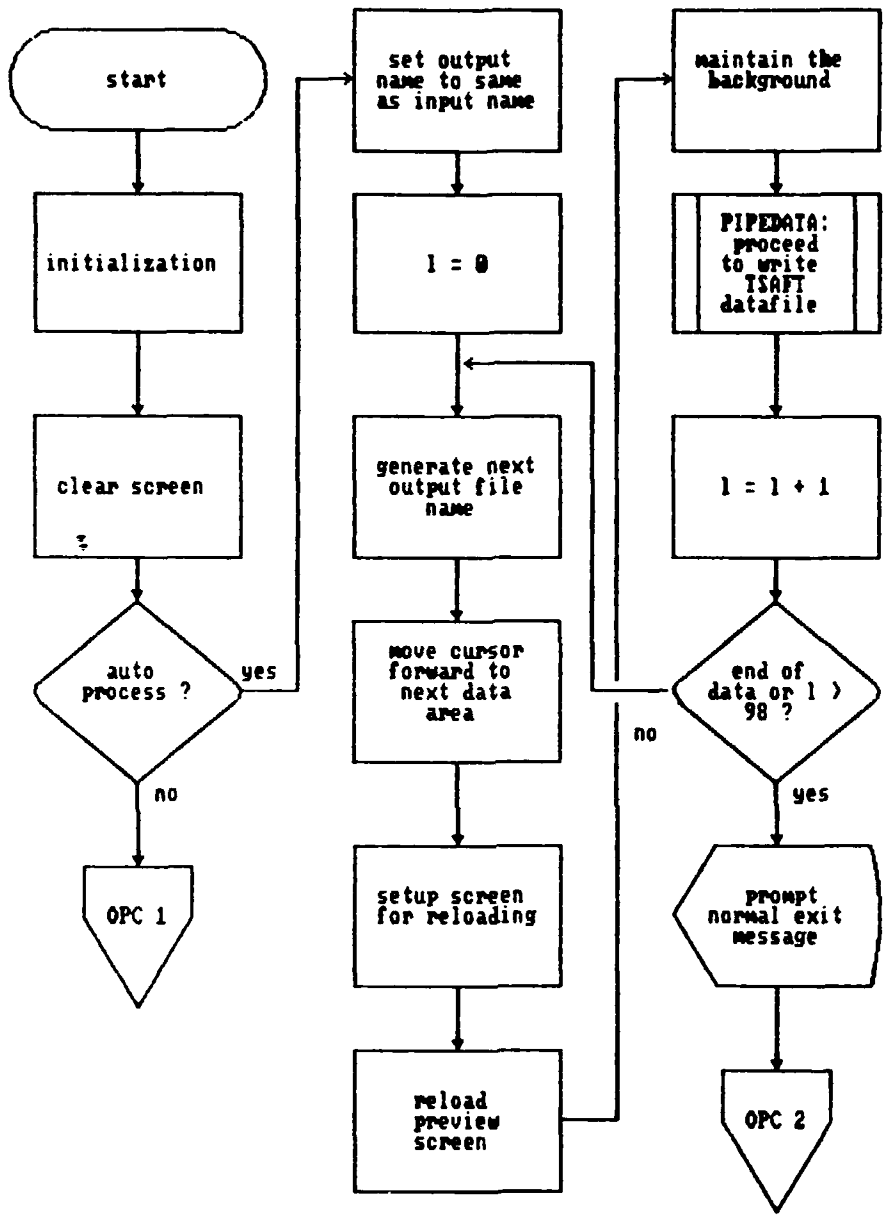

start
clear screen
SET_VAR: set all the variables from global memory
PARMINPUT2: read input output file names & other parameters
SETUP: set up screen scales for drawing
seek for A-scan delimiter
k = 1
k > traces ?
no
yes
close raw input data file
signal display completion
width calibration ?
yes
CALWIDTH: calibrate data to survey width
no
ANNOTATE: write pertinent parameters to screen
OPC 1
position start read to first scan byte
read one scan
plot one line
k = k + 1

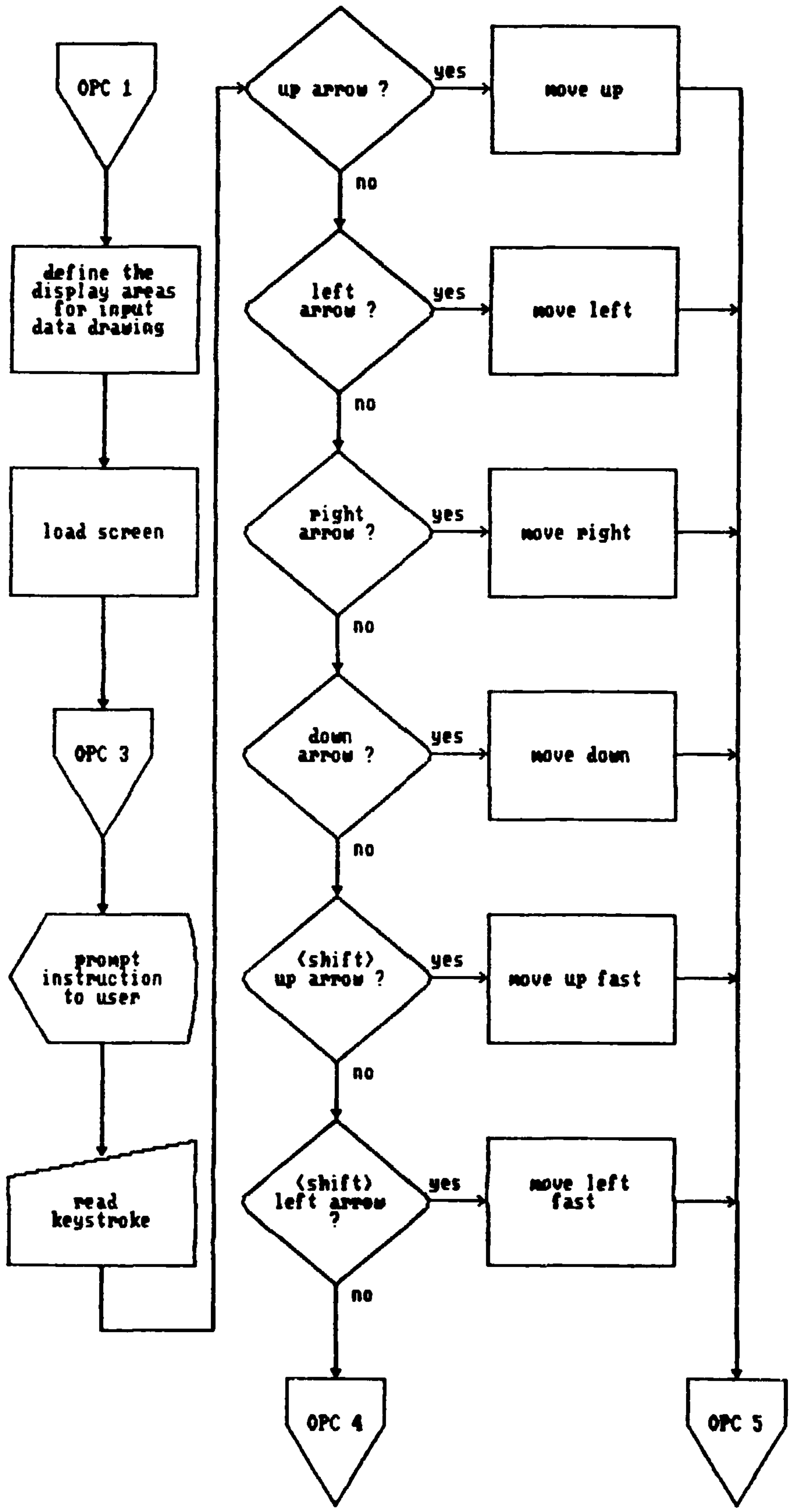

OPC 1
fname exist ?
no
yes
save screen info, write header, close Tsaft data file
print screen ?
no
yes
print screen
leave graphic
clear screen
set header to local variables value
autoprocess ?
no
yes
forward control to TRANSFER workhorse
return control to PREVIEW menu

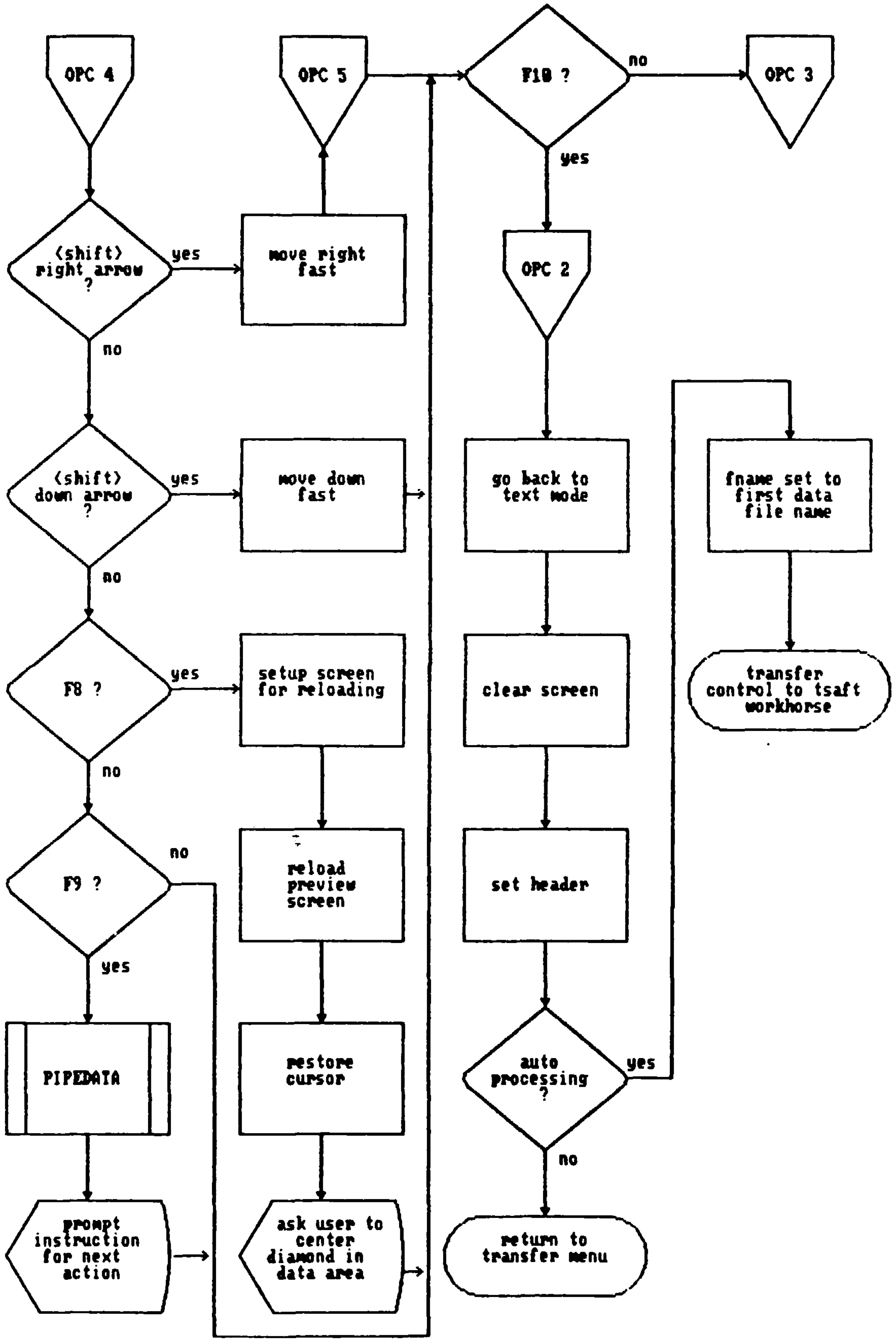

start
set initial values
restore original preview data screen
ask user to move diamond to left mark, then Press F1
read keystroke
up arrow ?
yes
move polygon up
no
left arrow ?
yes
move left
no
right arrow ?
yes
move right
no
down arrow ?
yes
move down
no
(shift) up arrow ?
yes
move up fast
no
OPC 1
OPC 2

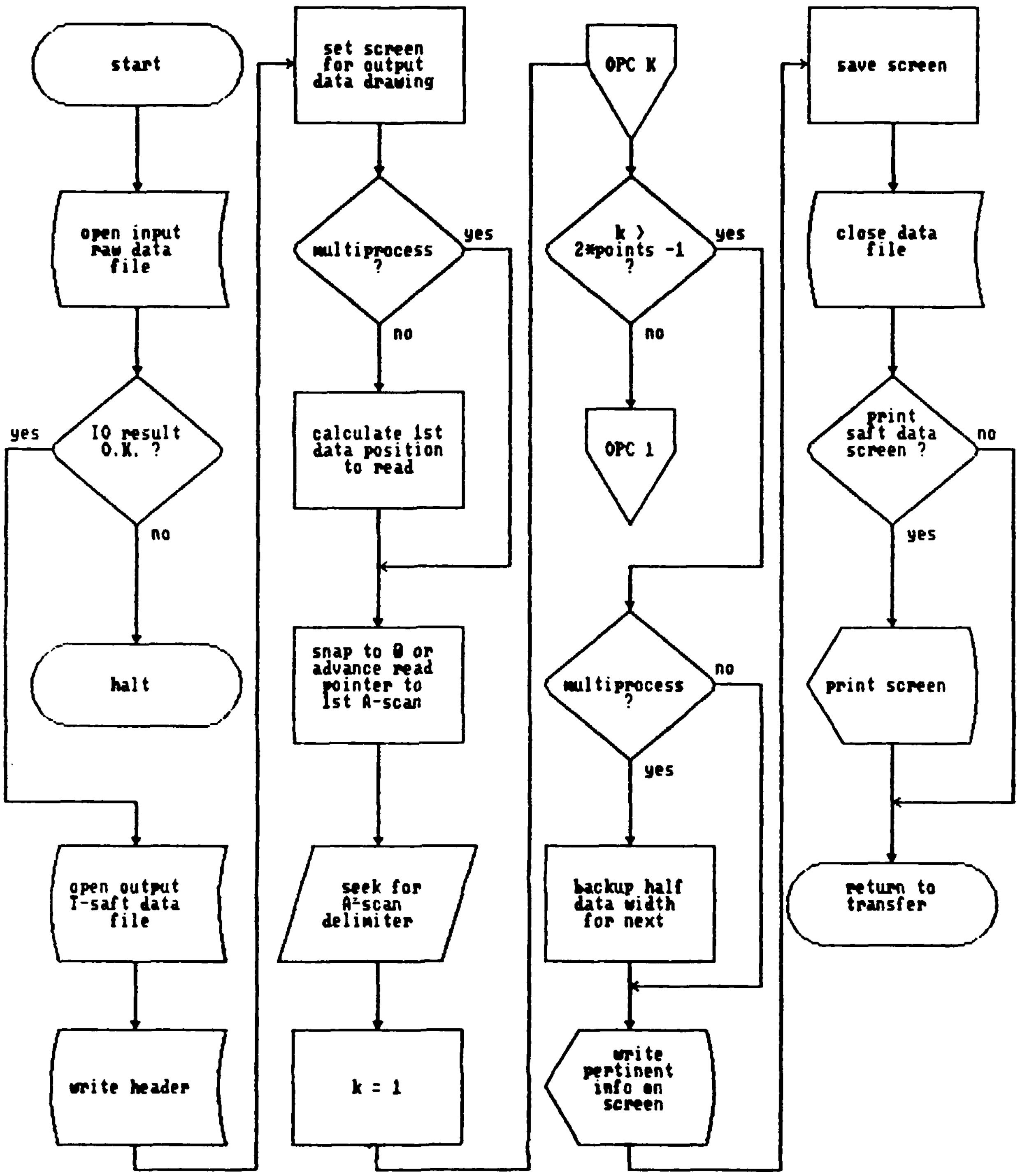


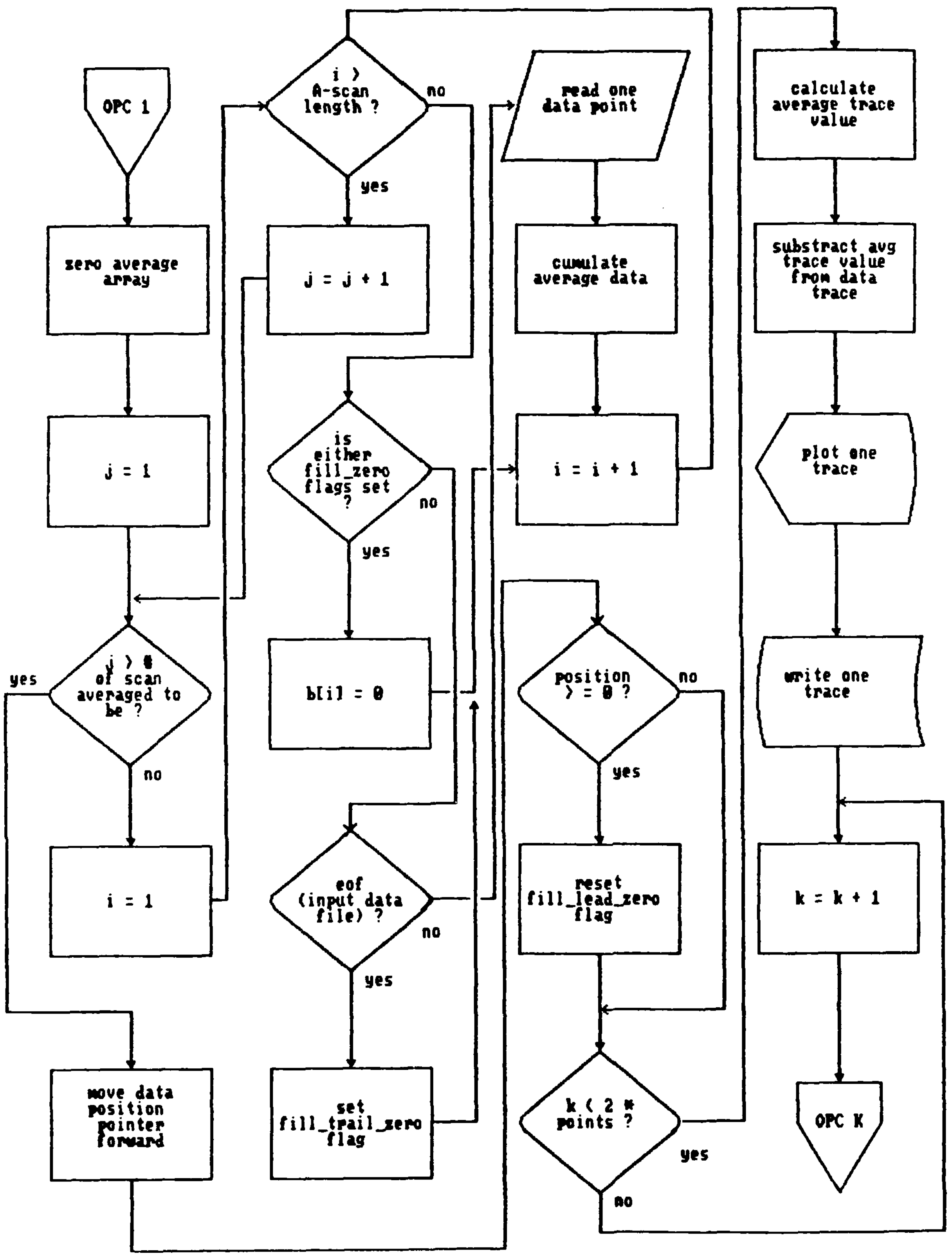
OPC 1
(shift) left arrow ?
yes
move left fast
OPC 2
no
(shift) right arrow ?
yes
move right fast
no
(shift) down arrow ?
yes
move down fast
no
F1 ?
yes
read cursor pos, ask user to move diamond to right mark, then Press F2
ask user to move diamond to station benchmark, then Press F3
no
F2 ?
yes
read cursr pos, read distance between marks
calculate spacing between averaged scans
no
F3 ?
yes
read station number input
draw station marker, display Press F4 to finish prompt
no
return to PREVIEW workhorse calling routine
yes
F4
no

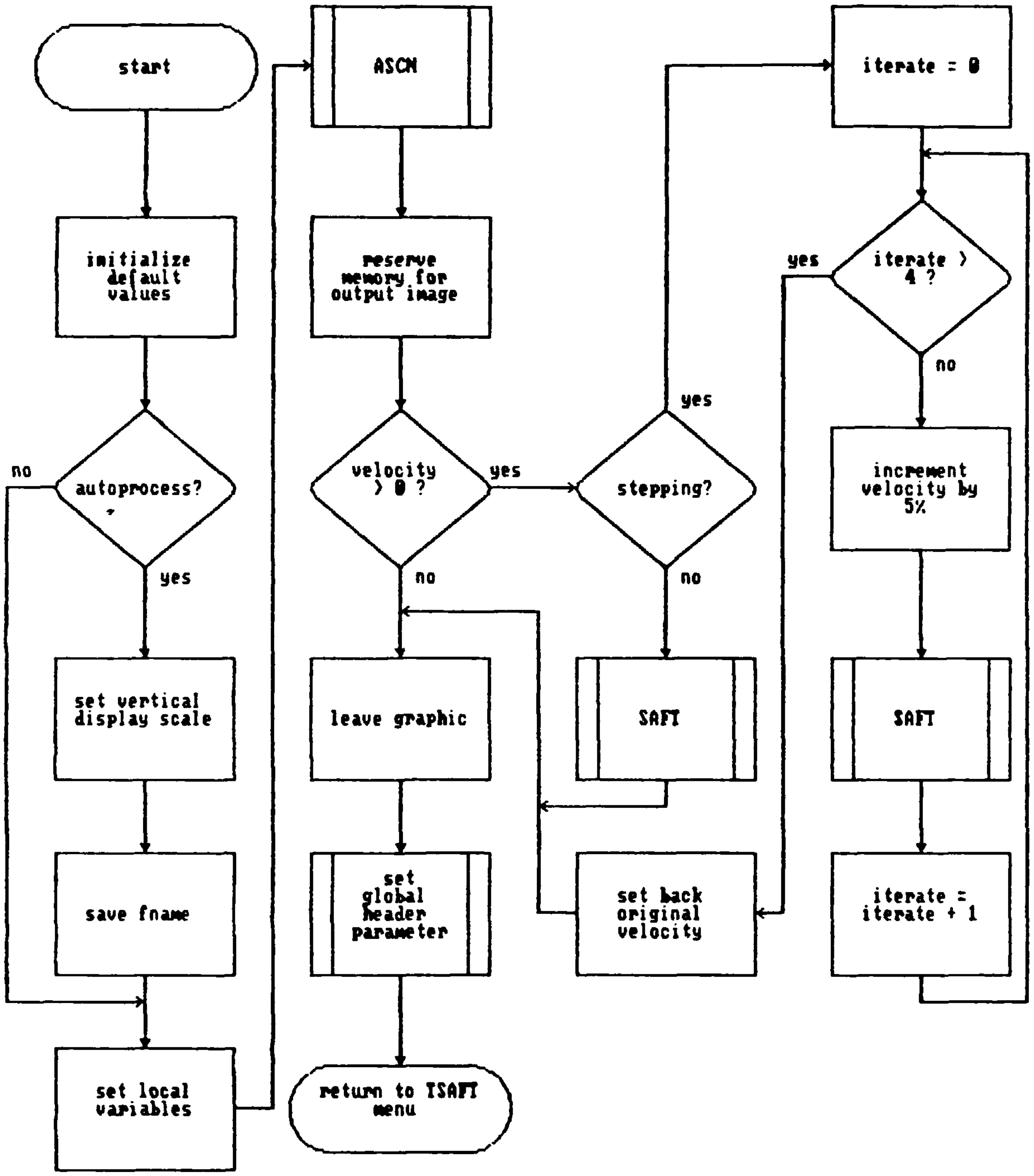

start
initialization
clear screen
auto process ?
yes
no
OPC 1
set output name to same as input name
I = 0
generate next output file name
move cursor forward to next data area
setup screen for reloading
reload preview screen
maintain the background
PIPEDATA: proceed to write TSAFT datafile
I = I + 1
end of data or I > 98 ?
no
yes
prompt normal exit message
OPC 2

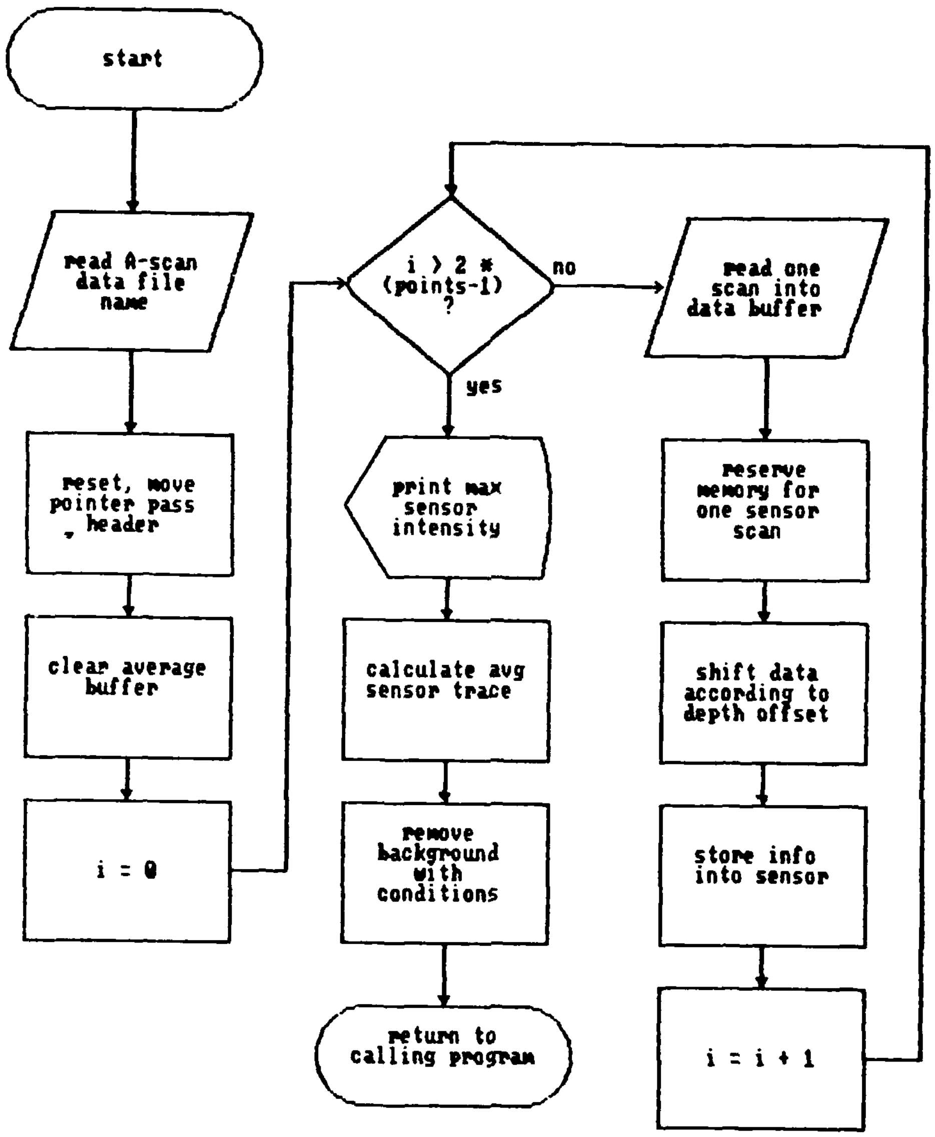

OPC 1
define the display areas for input data drawing
load screen
OPC 3
prompt instruction to user
read keystroke
up arrow ?
yes
move up
no
left arrow ?
yes
move left
no
right arrow ?
yes
move right
no
down arrow ?
yes
move down
no
(shift) up arrow ?
yes
move up fast
no
(shift) left arrow ?
yes
move left fast
no
OPC 4
OPC 5

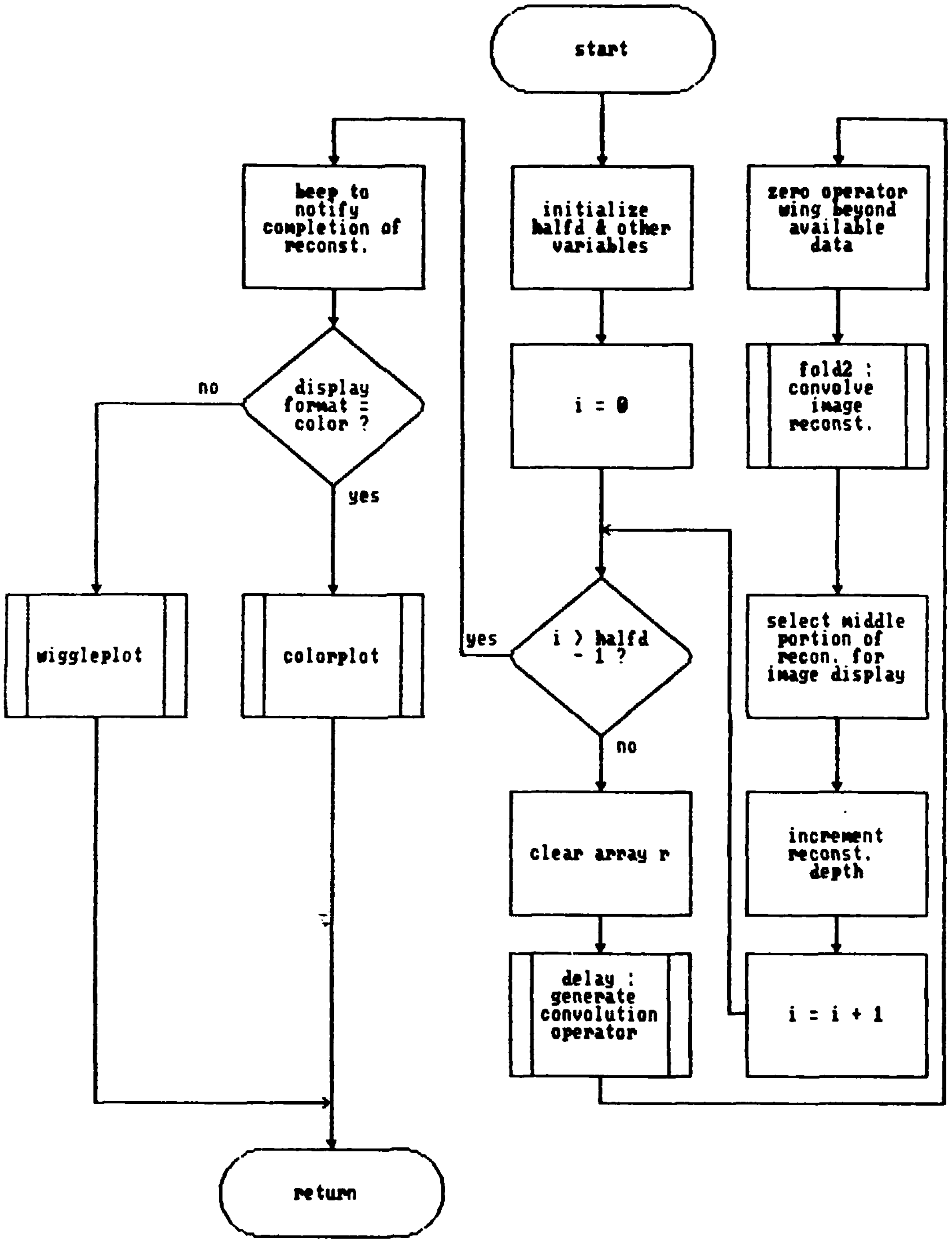

OPC 4
OPC 5
F10 ?
no
OPC 3
yes
<shift> right arrow ?
yes
move right fast
OPC 2
no
<shift> down arrow ?
yes
move down fast
go back to text mode
fname set to first data file name
no
F8 ?
yes
setup screen for reloading
clear screen
transfer control to tsaft workhorse
no
F9 ?
no
reload preview screen
set header
yes
PIPEDATA
restore cursor
auto processing ?
yes
no
prompt instruction for next action
ask user to center diamond in data area
return to transfer menu


start
open input raw data file
IO result O.K. ?
yes
no
halt
open output T-saft data file
write header
set screen for output data drawing
multiprocess ?
yes
no
calculate 1st data position to read
snap to 0 or advance read pointer to 1st A-scan
seek for A-scan delimiter
k = 1
OPC K
k > 2*points -1 ?
yes
no
OPC 1
multiprocess ?
no
yes
backup half data width for next
write pertinent info on screen
save screen
close data file
print saft data screen ?
no
yes
print screen
return to transfer

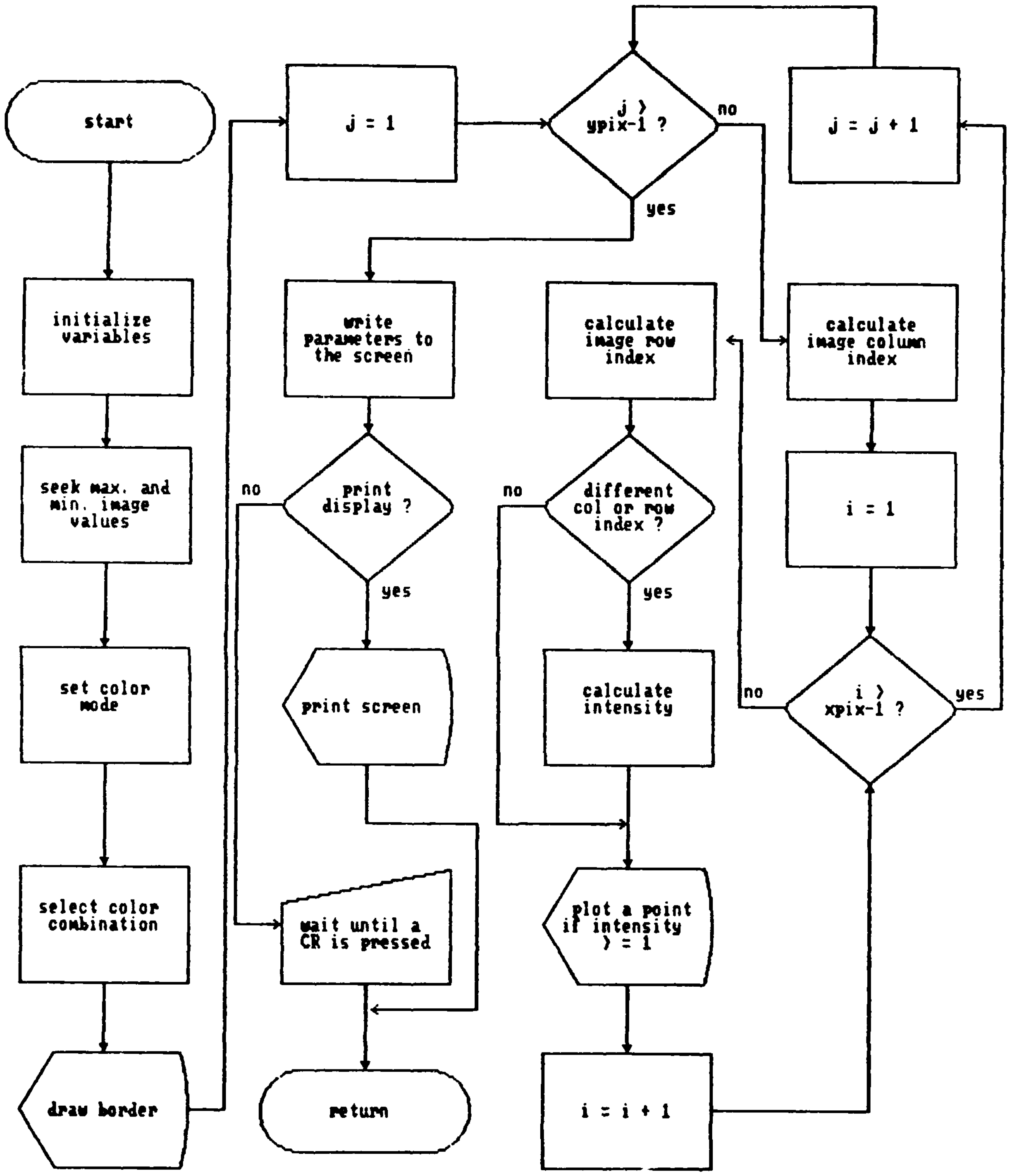

OPC 1
zero average array
j = 1
j > # of scan averaged to be ?
yes
no
i = 1
move data position pointer forward
i > A-scan length ?
no
yes
j = j + 1
is either fill_zero flags set ?
no
yes
b[i] = 0
eof (input data file) ?
no
yes
set fill_trail_zero flag
read one data point
cumulate average data
i = i + 1
position > = 0 ?
no
yes
reset fill_lead_zero flag
k < 2 * points ?
yes
no
calculate average trace value
substract avg trace value from data trace
plot one trace
write one trace
k = k + 1
OPC K

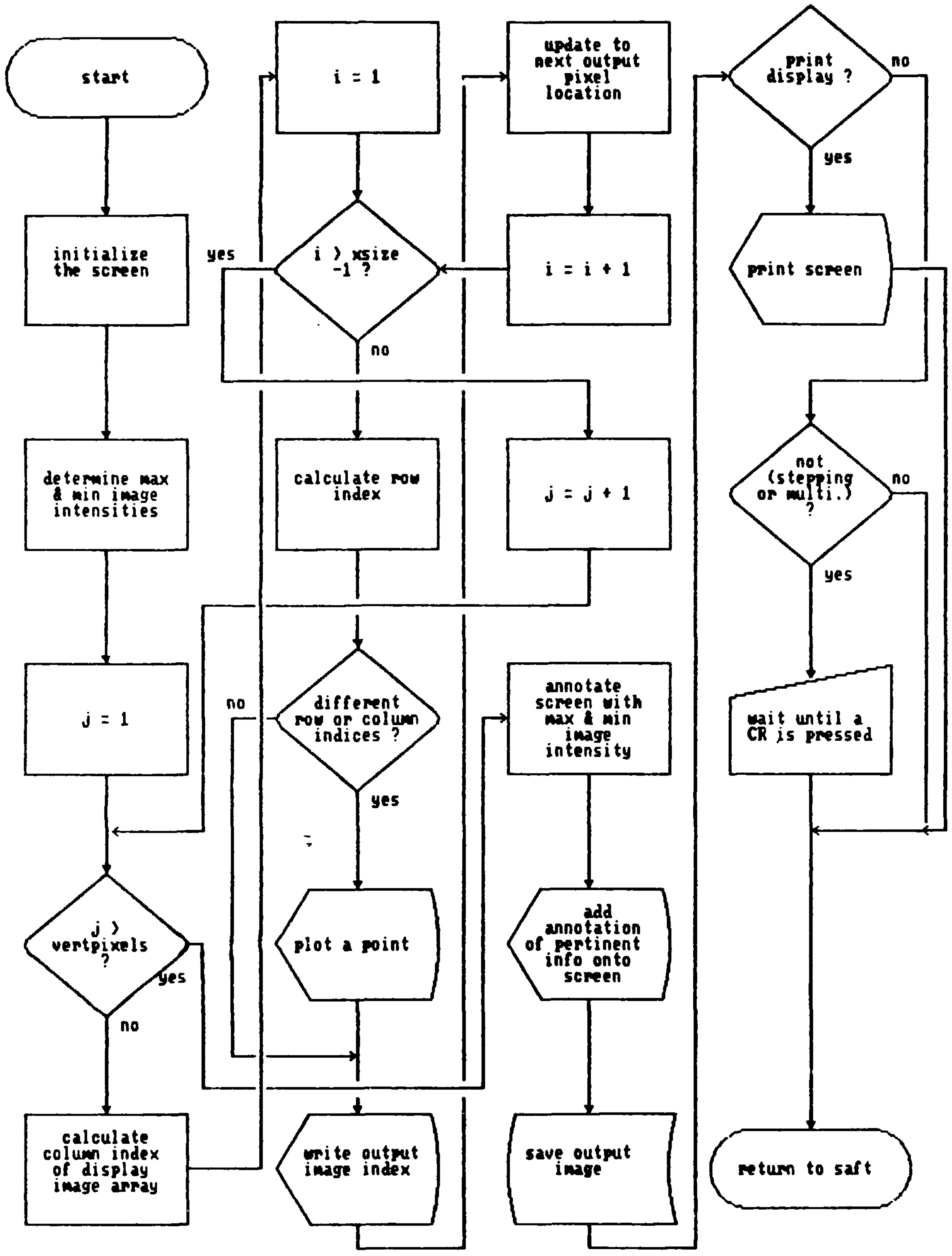

start
initialize default values
autoprocess?
no
yes
set vertical display scale
save fname
set local variables
ASCN
reserve memory for output image
velocity > 0 ?
yes
no
leave graphic
set global header parameter
return to TSAFT menu
stepping?
yes
no
SAFT
set back original velocity
iterate = 0
iterate > 4 ?
yes
no
increment velocity by 5%
SAFT
iterate = iterate + 1

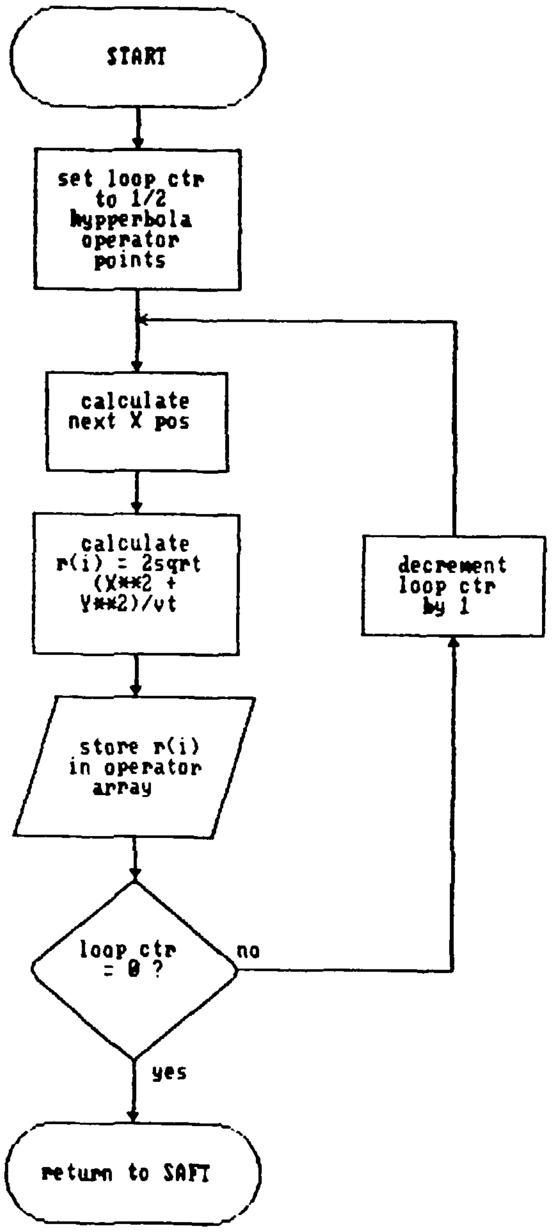

start
read A-scan data file name
reset, move pointer pass header
clear average buffer
i = 0
i > 2 * (points-1) ?
no
yes
read one scan into data buffer
reserve memory for one sensor scan
shift data according to depth offset
store info into sensor
i = i + 1
print max sensor intensity
calculate avg sensor trace
remove background with conditions
return to calling program

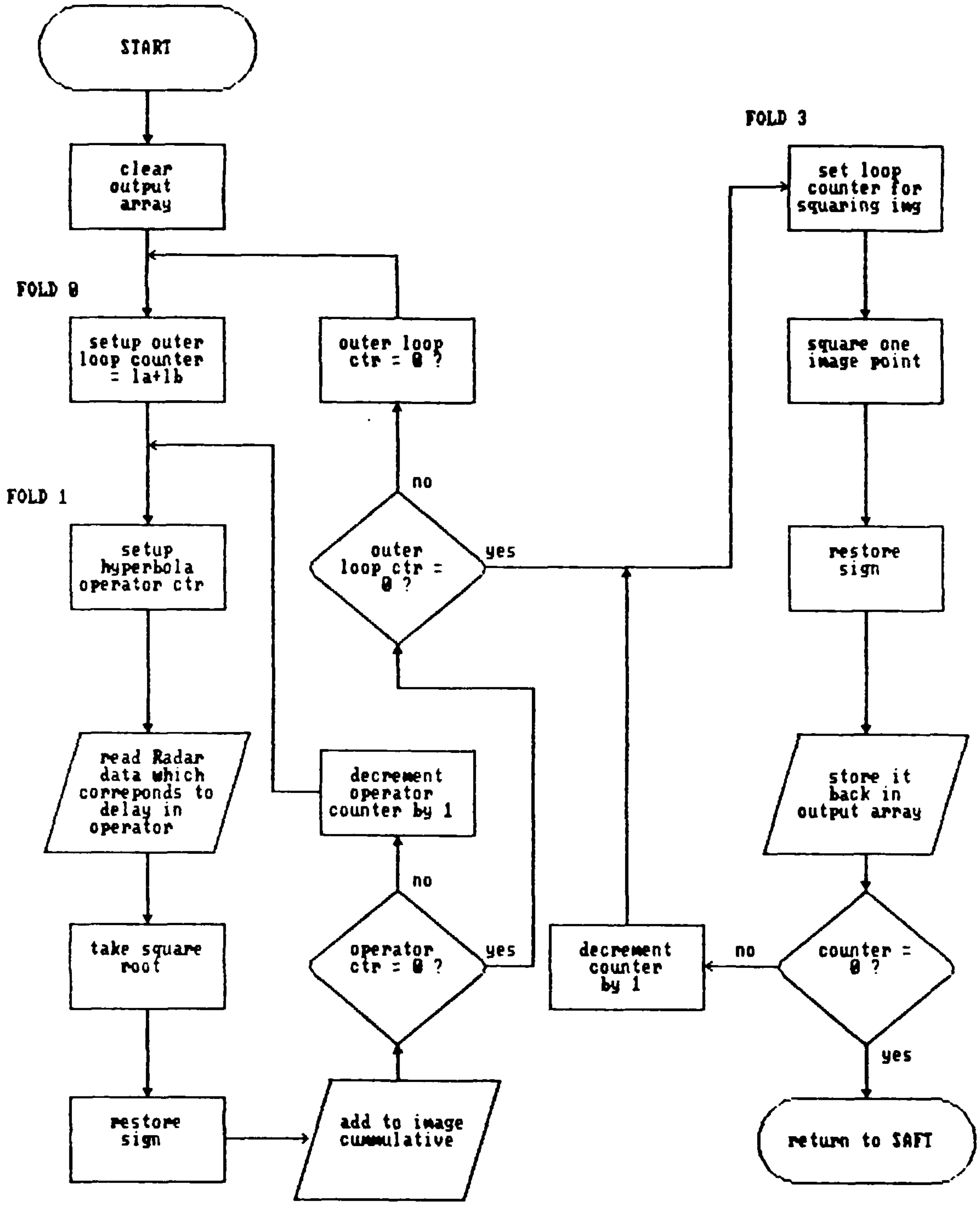

start
initialize halfd & other variables
i = 0
i > halfd - 1 ?
yes
no
clear array r
delay : generate convolution operator
zero operator wing beyond available data
fold2 : convolve image reconst.
select middle portion of recon. for image display
increment reconst. depth
i = i + 1
beep to notify completion of reconst.
display format = color ?
no
yes
wiggleplot
colorplot
return

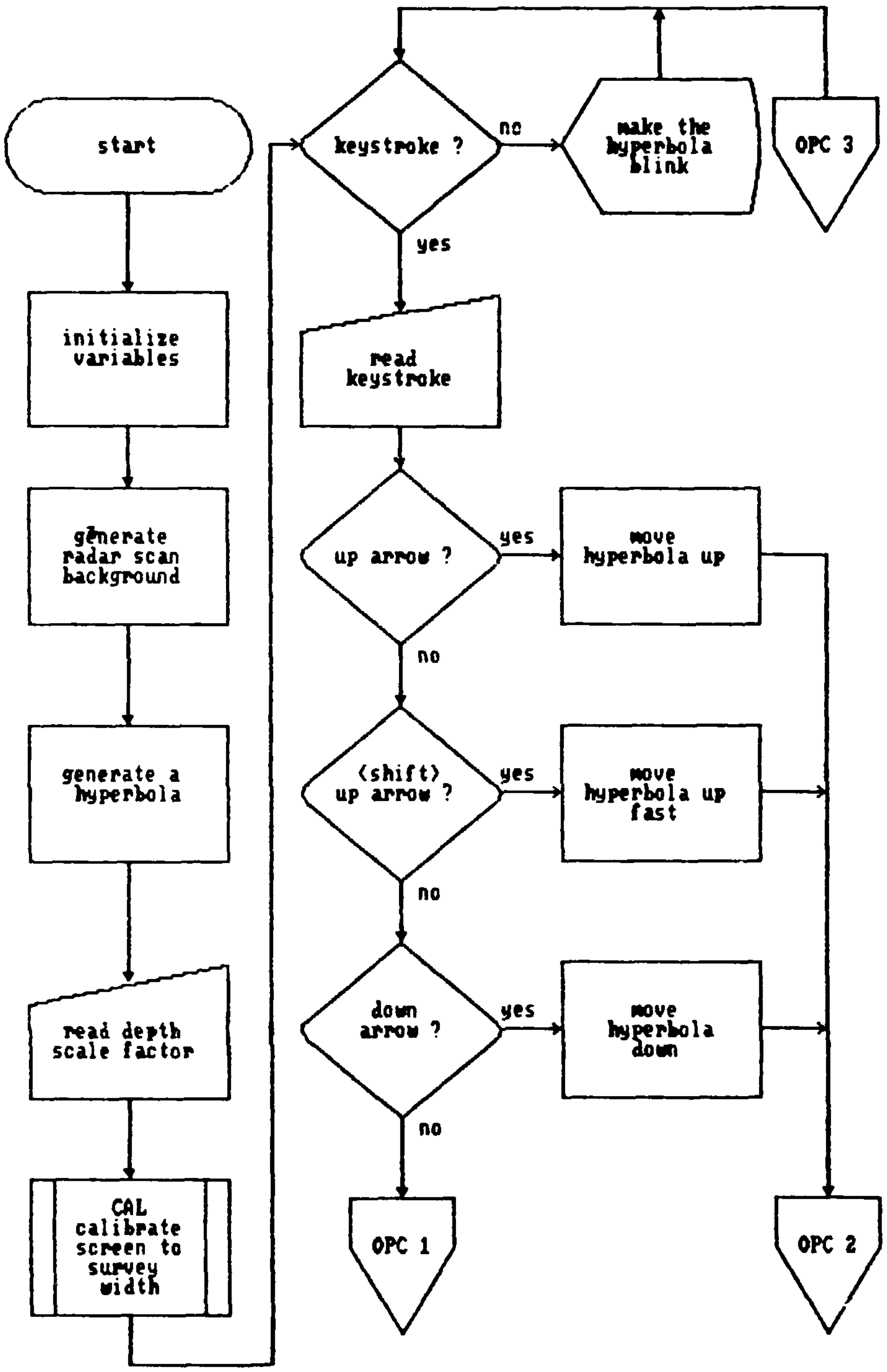
COLORPLOT -- TSAFT SUBROUTINE

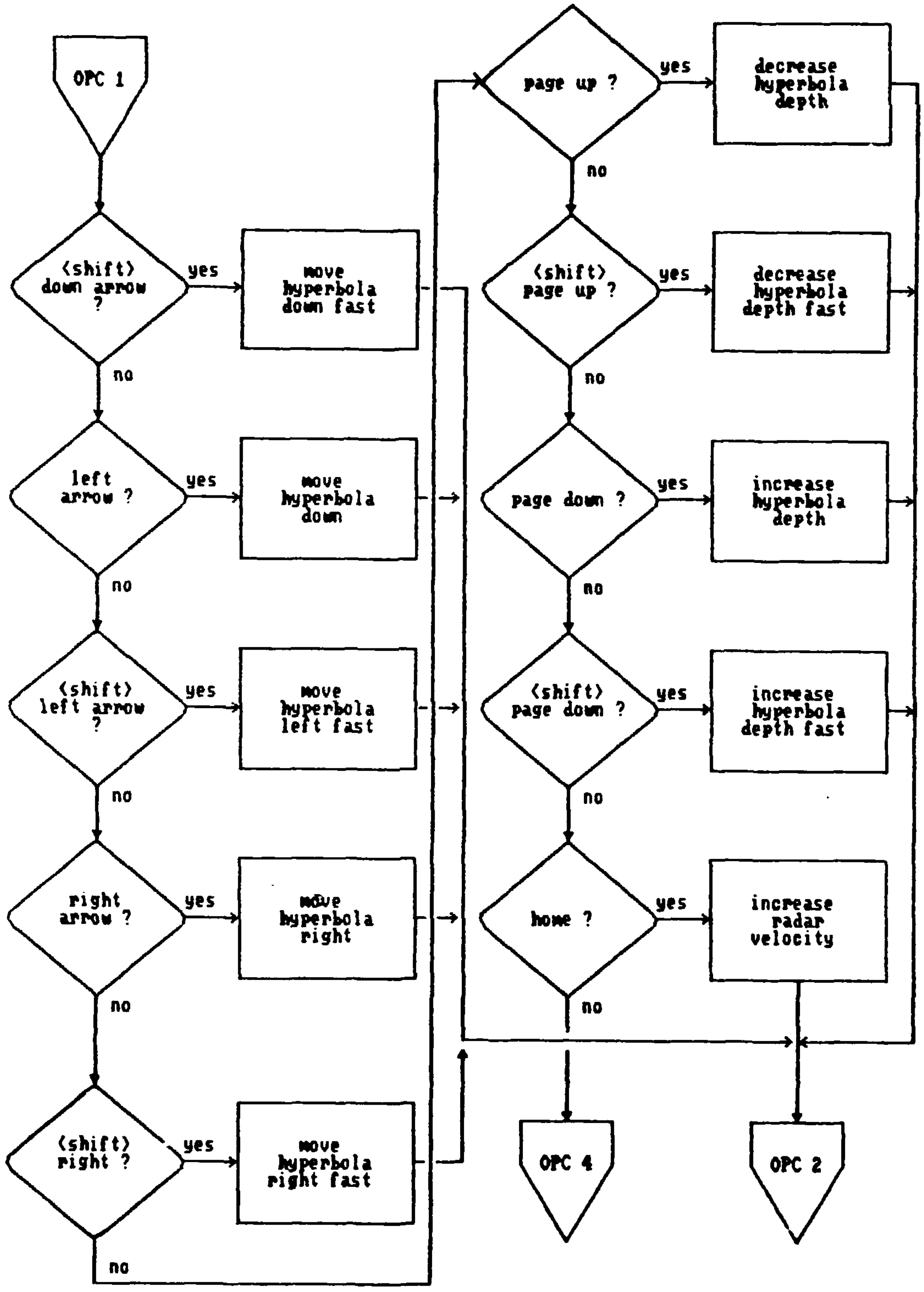

start
initialize variables
seek max. and min. image values
set color mode
select color combination
draw border
j = 1
j > ypix-1 ?
no
yes
write parameters to the screen
print display ?
no
yes
print screen
wait until a CR is pressed
return
calculate image row index
different col or row index ?
no
yes
calculate intensity
plot a point if intensity > = 1
i = i + 1
j = j + 1
calculate image column index
i = 1
i > xpix-1 ?
no
yes

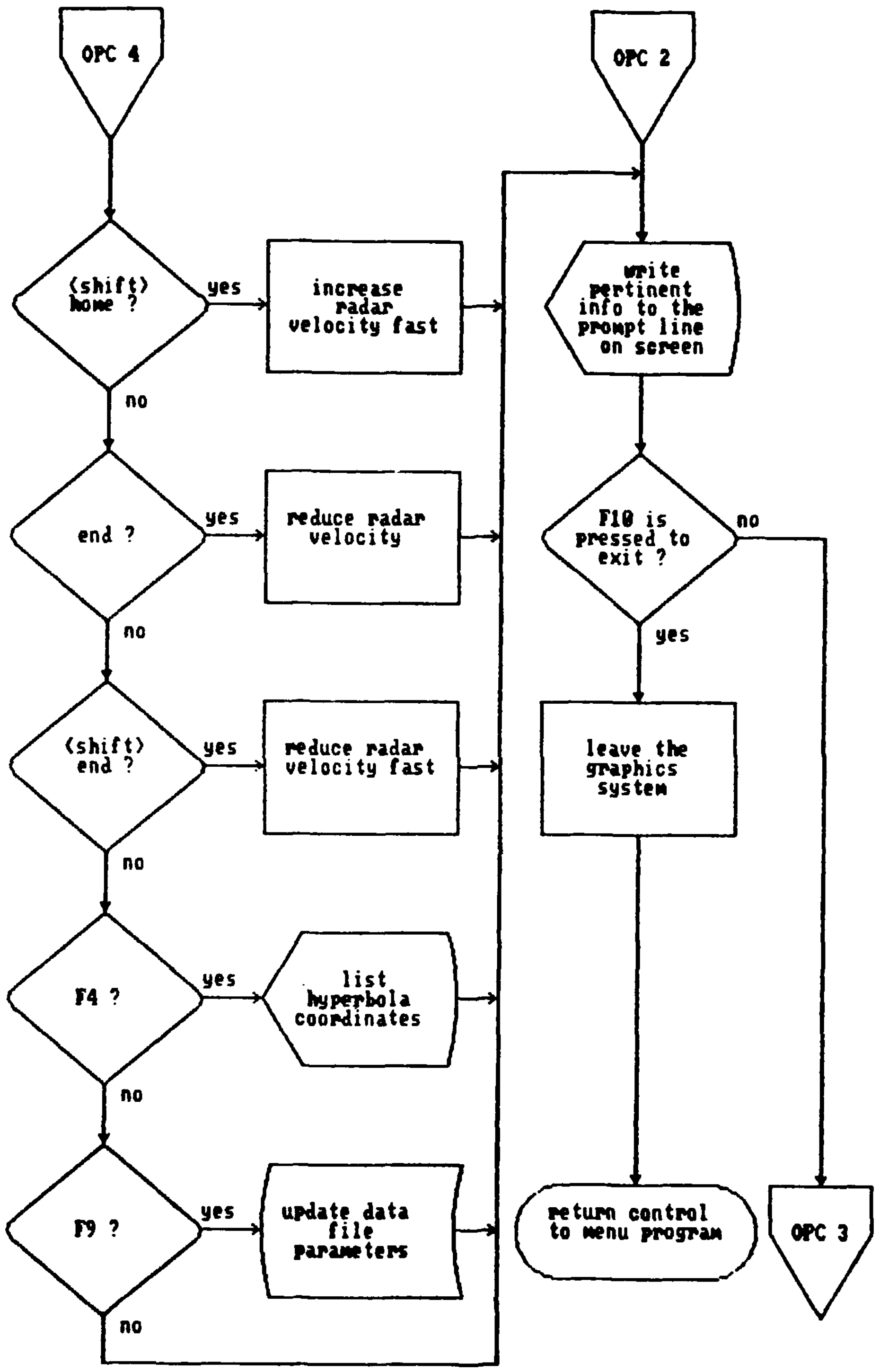

start
initialize the screen
determine max & min image intensities
j = 1
j > vertpixels ?
yes
no
calculate column index of display image array
i = 1
i > xsize -1 ?
yes
no
calculate row index
different row or column indices ?
no
yes
plot a point
write output image index
update to next output pixel location
i = i + 1
j = j + 1
annotate screen with max & min image intensity
add annotation of pertinent info onto screen
save output image
print display ?
no
yes
print screen
not (stepping or multi.) ?
no
yes
wait until a CR is pressed
return to saft

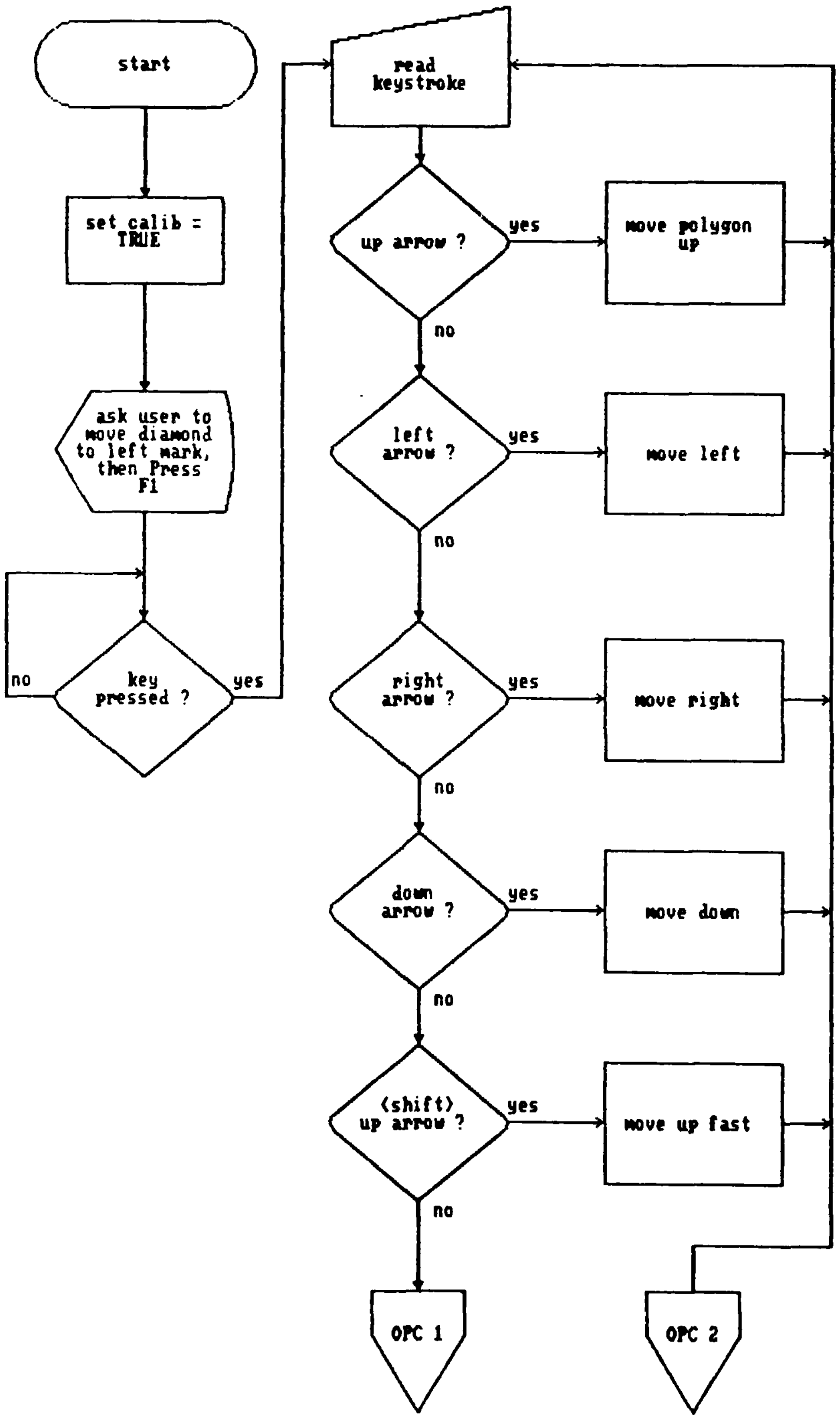

START
set loop ctr to 1/2 hypperbola operator points
calculate next X pos
calculate r(i) = 2sqrt (X**2 + Y**2)/vt
store r(i) in operator array
loop ctr = 0 ?
no
yes
decrement loop ctr by 1
return to SAFT

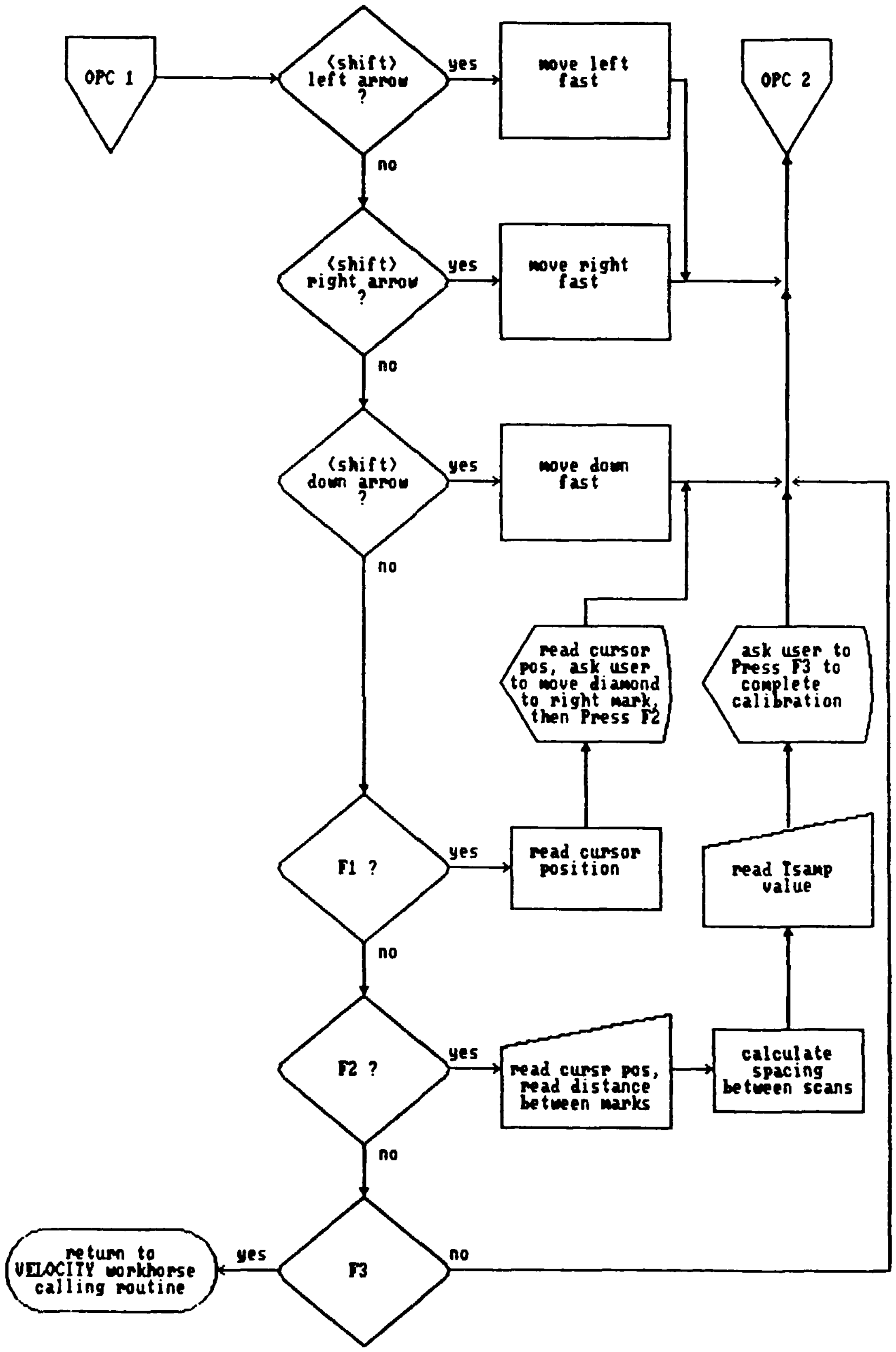

START
clear output array
FOLD 0
setup outer loop counter = la+lb
FOLD 1
setup hyperbola operator ctr
read Radar data which corrreponds to delay in operator
take square root
restore sign
add to image cummulative
operator ctr = 0 ?
no
yes
decrement operator counter by 1
outer loop ctr = 0 ?
no
yes
outer loop ctr = 0 ?
FOLD 3
set loop counter for squaring img
square one image point
restore sign
store it back in output array
counter = 0 ?
no
yes
decrement counter by 1
return to SAFT

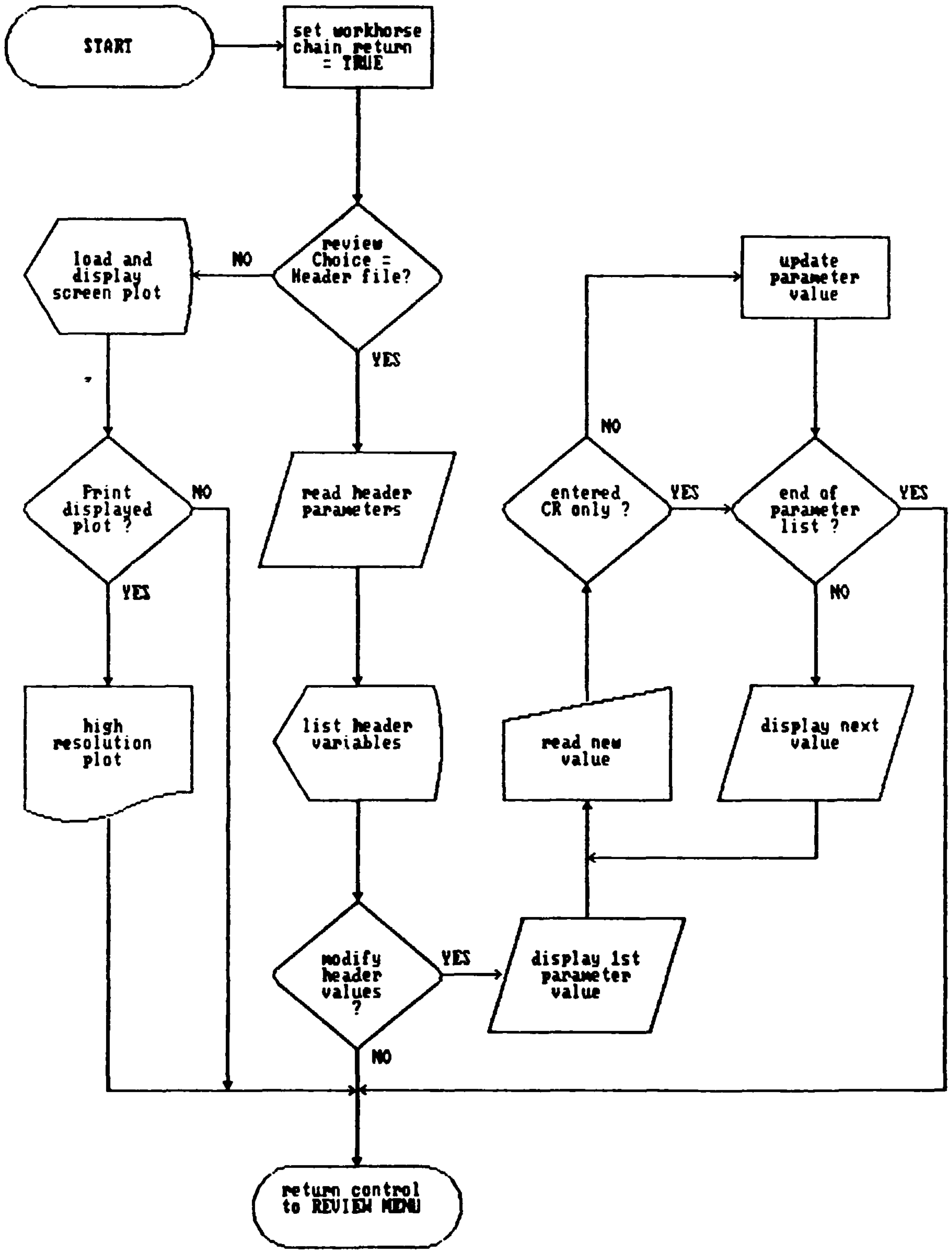

start
initialize variables
generate radar scan background
generate a hyperbola
read depth scale factor
CAL calibrate screen to survey width
keystroke ?
no
make the hyperbola blink
OPC 3
yes
read keystroke
up arrow ?
yes
move hyperbola up
no
(shift) up arrow ?
yes
move hyperbola up fast
no
down arrow ?
yes
move hyperbola down
no
OPC 1
OPC 2


OPC 1
<shift> down arrow ?
yes
move hyperbola down fast
no
left arrow ?
yes
move hyperbola down
no
<shift> left arrow ?
yes
move hyperbola left fast
no
right arrow ?
yes
move hyperbola right
no
<shift> right ?
yes
move hyperbola right fast
no
page up ?
yes
decrease hyperbola depth
no
<shift> page up ?
yes
decrease hyperbola depth fast
no
page down ?
yes
increase hyperbola depth
no
<shift> page down ?
yes
increase hyperbola depth fast
no
home ?
yes
increase radar velocity
no
OPC 4
OPC 2


OPC 4
(shift) home ?
yes
increase radar velocity fast
no
end ?
yes
reduce radar velocity
no
(shift) end ?
yes
reduce radar velocity fast
no
F4 ?
yes
list hyperbola coordinates
no
F9 ?
yes
update data file parameters
no
OPC 2
write pertinent info to the prompt line on screen
F10 is pressed to exit ?
no
yes
leave the graphics system
return control to menu program
OPC 3


start
set calib = TRUE
ask user to move diamond to left mark, then Press F1
key pressed ?
no
yes
read keystroke
up arrow ?
yes
move polygon up
no
left arrow ?
yes
move left
no
right arrow ?
yes
move right
no
down arrow ?
yes
move down
no
(shift) up arrow ?
yes
move up fast
no
OPC 1
OPC 2


OPC 1
(shift) left arrow ?
yes
move left fast
OPC 2
no
(shift) right arrow ?
yes
move right fast
no
(shift) down arrow ?
yes
move down fast
no
read cursor pos, ask user to move diamond to right mark, then Press F2
ask user to Press F3 to complete calibration
F1 ?
yes
read cursor position
read Tsamp value
no
F2 ?
yes
read cursr pos, read distance between marks
calculate spacing between scans
no
return to VELOCITY workhorse calling routine
yes
F3
no


START
set workhorse chain return = TRUE
review Choice = Header file?
NO
YES
load and display screen plot
Print displayed plot ?
NO
YES
high resolution plot
read header parameters
list header variables
modify header values ?
YES
NO
display 1st parameter value
read new value
entered CR only ?
NO
YES
update parameter value
end of parameter list ?
YES
NO
display next value
return control to REVIEW MENU

5.0 NCEL RADAR IMAGING SOFTWARE LISTINGS

Menu Flow Listings

NCEL Title
SELECT
AUTOPROC/PREVIEW
TRANSFER
T-SAFT
VELOCITY
REVIEW

Working Program Listings

PREVIEW
TRANSFER
VELOCITY
REVIEW

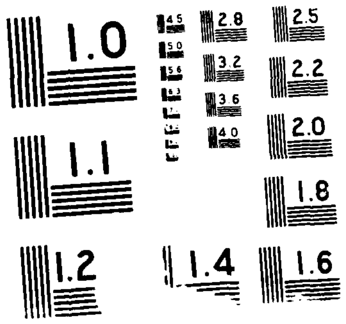
```
(************************* NCEL TITLE MENU PROGRAM *****************************)
(*                                                                             *)
(*     This menu program has been generated by a  screen utility program      *)
(*  listed below. Modifications have been made of this program to add global *)
(*  variable definitions, variables value setting and resetting routines,    *)
(*  input checking codes and additional program flow software.  These        *)
(*  enhancements are made in all the menu programs to different degrees      *)
(*  depending on specific requirements.  This program has been well          *)
(*  documented here as well as in the screen sculptor manuals.               *)
(*                                                                             *)
(*     The function of this menu program is to provide a friendly user        *)
(*  interface for program control and parameter input. when this program is *)
(*  executed predetermined values are given to these varisbles.  The         *)
(*  operator can change these values and is provided immediate feed back     *)
(*  on the screen.  In this program values presented on the screen takes     *)
(*  precedence over the program header values.  Displayed values can be      *)
(*  updated by simply depressing the carriage return when the cursor is at   *)
(*  the input file name.                                                      *)
(*                                                                             *)
(*                                                             7-1-87          *)
(*******************************************************************************)

PROGRAM TITLE(INPUT,OUTPUT);
{Procedures Generated By Screen Sculptor,Version 1.2 5/17/1987 - 15:35:46 }
CONST CopyrightSS='(C)Copyright 84,85 The Software Bottling Company Of New York';
{    DO NOT REMOVE The Above Copyright Notice
     This Program may not be used without the above Copyright Notice
}
{  SCREEN SCULPTOR(C) Version 1.2
   (C) COPYRIGHT, THE SOFTWARE BOTTLING COMPANY OF NEW YORK, 1984, 1985
   WARNING: Do not attempt to make any changes to this procedure unless you
   have made a backup. The Software Bottling Company Of New York can not
   and will not support such changes made to this program.
   Turbo(tm) Pascal Version, Trade Mark Of Borland International}

{ Global Required Definition of variables and constants }


    TYPE                              {writeheader variables}
          id = RECORD
                 nx,                  {nx= # of A-scans in one data set}
                 nz     :integer;     {nx = A-scan lenght, normally 400}
                 w,                   {w = survey width}
                 dt,                  {dt = sampling period}
                 v      :real;        {v = radar propagation velecity}
                 sp     :real;        {sp = averaged data scan-scan spacing}
                 Xavg   :integer;     {Xavg = # of A-scan averaged per data A-scan}
                 stn    :integer;     {station = data station}
                 ln     :real;        {ln = data lenght}
                 scns   :integer;     {scns =# of A-scansin the data}
                 mrk    :integer;     {delimiter between A-scans in original data}
                 zoffset:integer;     {template determined sample offset}
                 dcap   :real;        { data capacity of a disk }
                 srcfile:string[40];{scource data file}
```

```
                spare1:real;      {template determined velocity}
                spare2:integer;   {spare}
              END;
    VAR   { global variables }
          fname: string[40];        {name of I/O file}
          fin: string[40];          {name of input file}
          workhorse:file;           {chainfile assignee}
          fout:string[40];          {name of output file}
          which_screen:integer;     {which screen to use }
          chain_return:boolean;     {returning from chain program?}
          auto_processing:string[14];{automatic disk processing key}
          {the following characters are used for program flow control}
          char1,                    {plot intensity var in Preview}
          char2,                    {width calibtation var in Preview}
          char3,                    {print var in Preview}
          char4,                    {print var in Transfer}
          char5,                    {disk process var in Transfer}
          char6,                    {multiple processing var in T-saft}
          char7,                    {velocity stepping var in T-saft}
          char8,                    {print image var in T-saft}
          char9,                    {display format var in T-saft}
          char10,              {spare character variables}
          char11,              {spare character variables}
          char12,              {spare character variables}
          char13,              {spare character variables}
          char14,              {spare character variables}
          char15:char;         {spare character variables}
          sparename1,
          sparename2,
          sparename3,
          sparename4:string[40];    {spare strings}
          spareI1,
          spareI2,
          spareI3,
          spare14:integer;          {spare integers}
          spareR1,
          spareR2,
          spareR3,
          spareR4:real;             {spare reals}
          info:id;                  {header information}

{the above are global variables to be passed from progran to program during
 the chain calls. These variables are to be declared exactly as above in all
 programs wether used or not.  Tailoring will be done with the spares.}


{what follows below are translation of the above header variables}

 VAR  points,aslen:real{integer};
      width,tsamp,vel,calspace:real;
      navg,station:real{integer};
      lenght:real;
      scans,mark:real{integer};
      spare:real;
```

```
            datcap:real;
            srcfile:string[14];
            detv:real;
            depthoff:real{integer};
            datafilenames1,datafilenames2:string[14];
            datfile:file;
            buff: ARRAY[0..127] OF byte;  {input buffer for blocking data and header}
            OK:boolean;                  {file existence flag}
            indexinc:integer;            {file index increment in multiproc}


      TYPE  STR2 = STRING[2]; STR80 = STRING[80]; STR79 = STRING[79];
            resSS = (staySS, prevSS, exitSS, nextSS);

      CONST { Esc, Up Arrow Key, Left Arrow Key , Page Up Key       }
            escSS=#27;   uSS='H'; lSS='K'; puSS='I';
            { Blank, Down Arrow Key, Right Arrow Key, Page Down Key }
            blankSS=' '; dSS='P'; rSS='M'; pdSS='Q';
            { Function keys  F1-F10 }
            f1SS=';'; f2SS='<'; f3SS='='; f4SS='>'; f5SS='?';
            f6SS=' '; f7SS='A'; f8SS='B'; f9SS='C'; f10SS='D';
            retSS : STR2='';

      VAR   actionSS, last_field_actionSS : resSS;
            hiSS, loSS : REAL;
            vtypeSS, screenSS, use_screenSS, screen_fieldSS, varSS : INTEGER;
            file_existSS, last_fieldSS, retrieveSS : BOOLEAN;
            rangeSS : STR80;
            BeepOnSS: BOOLEAN;

        {$V-,C-,R-} { Pascal Directives. See Compiler Manual }
        {$I TURPROCS.TUR  Include Procedures In This File. See Manual }

      (*   This is a summary of the procedures in TURPROCS.TUR

      PROCEDURE BEEP(BeepOn: BOOLEAN);              { Sound Beep if BeepOn=TRUE }
      PROCEDURE CLEAR_KBD;                          { Clear Keyboard Buffer }
      PROCEDURE COLOR(foregr,backgr:BYTE);          { Set Color }
      PROCEDURE WRITEC(vtext: STR80);               { Write Chars Using Color }
      FUNCTION  SET_MONITOR_TYPE: INTEGER;          { Determine Monitor Type }
      { Display A Screen Sculptor Screen }          { 2=Color, 3=Mono }
      PROCEDURE DISPLAY_SCREEN(screen_name: STR80; VAR file_existSS: BOOLEAN);
      { Display And Get An Item From Screen. See Detailed Desription In Manual }
      PROCEDURE GETITEM(
                      COL,LIN,LEN :         BYTE;      { Column, Line, Length }
                      ITYPE :               CHAR;      { Type= C, N, D, Y, M  }
                  VAR WITEM :               STR80;     { Variable Name        }
                      PICT :                STR80;     { Picture X, U, L, 9, 8 # }
                      ITEM_LOW,ITEM_HIGH :  STR80;     { Range - Numerics/Date Only}
                  VAR RET :                 STR2;      { Returned Code        }
                      RETRIEVE :            BOOLEAN;   { False=Disp Only, True=Get }
                      FGR_COLOR,BGR_COLOR : BYTE       { Colors Foregr, Backgr }
                      ); EXTERN;
```

```
*)

PROCEDURE RET_STATUS;
{ Check Status Of Variable retSS and return a code in 'actionSS'   set 'varSS'
  This procedure is called immediately following GETITEM }

{ Input to this procedure:
  when retSS is length 1 the values are any of the ASCII chars
  when retSS is length 2 the values are uSS, lSS, puSS, pdSS, function keys
                                        dSS, rSS
                                        ( See CONST Section For Meanings ) }
{ Output:
  The following codes are returned in actionSS : nextSS, prevSS,
                                                 exitSS, staySS }
{ Based upon 'actionSS' this procedure will then set 'varSS' to an integer,
  which represents the next item (variable ) to get.  }
BEGIN
  last_field_actionSS:=staySS;
  actionSS:=nextSS; { Initialize Action Code }
  if retrieveSS then { Is retrieveSS TRUE? }
  begin
    if ord(retSS[0])=2 then { Is retSS length 2 ? }
    begin
      CASE retSS[2] of
      { Action to be taken depending on the last key pressed }
        uSS, lSS: actionSS:=prevSS;  { Up Key, Left Key }
        dSS, rSS: actionSS:=nextSS;  {Down Key, Right Key}
        puSS: actionSS:= { Page Up   }staySS;
        pdSS: actionSS:= { Page Down }staySS;
        f1SS: actionSS:= {execute} exitSS;          (**** exits screen ****)
        f2SS,f3SS,f4SS,f5SS,
        f6SS,f7SS,f8SS,f9SS,f10SS: actionSS:=staySS; { Function Key }
      END { Case ret };
    end else { retSS is length 1 }
    begin
      if retSS=escSS { Escape Key } then actionSS:=exitSS
    end;
  { Any other key not in the above list will keep actionSS=nextSS }
  end; {retrieveSS}
  CASE actionSS of
    staySS: ;
    nextSS: begin
              varSS:=varSS+1; if varSS>screen_fieldSS then varSS:=1;
              if last_fieldSS and retrieveSS then actionSS:=last_field_actionSS
            end;
    prevSS: begin varSS:=varSS-1; if varSS<1 then varSS:=screen_fieldSS end;
    exitSS: ;
  END; { CASE }
END; {PROCEDURE RET_STATUS}


PROCEDURE GETNUM(
                COL,LIN,LEN :          BYTE;      { Column, Line, Length }
                ITYPE :                CHAR;      { Type= C, N, D, Y, M  }
                VAR WITEM :                 REAL;      { Numerci Variable Name }
                    PICT :                  STR80;     { Picture X, U, L, 9, 8 # }
                    ITEM_LOW,ITEM_HIGH :    REAL;      { Range - Numerics/Date Only}
                VAR RET :                   STR2;      { Returned Code          }
                    RETRIEVE :              BOOLEAN;   { False=Disp Only, True=Get }
                    FGR_COLOR,BGR_COLOR :   BYTE       { Colors Foregr, Backgr }
                    );
{ This Procedure converts numeric to string before calling GETITEM }
{ It then converts the result back to numeric }
VAR numSS,numloSS,numhiSS: STR80; errorcodeSS,dec_posSS: INTEGER;
BEGIN
  { Get # of Decimal Positions }
  dec_posSS:=ord(pict[0])-pos('.',pict);
  { Convert item, low and high range to string }
  STR(witem:0:dec_posSS,numSS);
  STR(item_low:0:dec_posSS,numloSS);
  STR(item_high:0:dec_posSS,numhiSS);
  GETITEM(col,lin,len,itype,numSS,pict,numloSS,numhiSS,
          ret,retrieve,fgr_color,bgr_color);
  { Convert string to numeric item }
  VAL(numSS, witem, errorcodeSS);
END; { GETNUM }


VAR { Variables Section For B:TITLE }
   S1_V1: STRING[1];

PROCEDURE INIT_VAR_1; { B:TITLE }
{ Initialize Variables To Default Values }
BEGIN
    S1_V1:='';
END; { PROCEDURE INIT_VARS_1}

PROCEDURE SCREEN_1; { B:TITLE }
BEGIN

if screenSS<>1 then
begin
  screenSS:=1; screen_fieldSS:=1; varSS:=1;
  retrieveSS:=FALSE; last_fieldSS:=FALSE;
  DISPLAY_SCREEN('TITLE.SCR',file_existSS);  { Display Screen }
  if not file_existSS then
  begin gotoxy(1,1); write('Missing Screen TITLE') end;
end;
retSS:='';

{ Display Items. Change retrieveSS to TRUE and INPUT items}
REPEAT { until actionSS = exitSS }
  CASE varSS of
1:  GETITEM(78,23,1,'C',S1_V1,
      'X','','',retSS,retrieveSS,0,0);
  END; { CASE }

  if varSS=screen_fieldSS then last_fieldSS:=TRUE;
```

AD-A194 286   SIGNAL PROCESSING SOFTWARE FOR GROUND PENETRATING RADAR   2/3
USER'S MANAAL(U) SIGMA RESEARCH INC REDMOND WA
R LIEM ET AL MAR 88 NCEL-CR-88-010 N62474-85-C-4953
UNCLASSIFIED   F/G 17/9   NL


1.0
4.5
5.0
5.6
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.2
1.4
1.6

```
  RET_STATUS; { Check the code in  retSS . Set   varSS  and  actionSS  }

{ Check to see whether to switch retrieveSS to true }
if last_fieldSS and (not retrieveSS) then
begin
  retrieveSS:=TRUE; last_fieldSS:=FALSE; actionSS:=staySS; varSS:=1;
end else
  last_fieldSS:=FALSE;

UNTIL actionSS=exitSS
END; { PROCEDURE SCREEN_1 }

PROCEDURE UPDOWN;
{ Will set 'use_screenSS' to the next screen to display }
{ Depending on the key pressed at the prompt.
  If key pressed is Page Up or Page Down then the 'use_screenSS' is
    incremented or decremented.
  If the key pressed is Esc then 'use_screenSS' is set to 0.
  If any other key pressed then 'use_screenSS' remains the same
    but 'retSS' is set to an Up Arrow Key.(uSS) }
VAR answerSS: STRING[1]; blank_lineSS: STRING[76];
BEGIN
  color(7,0); gotoxy(1,25);
  { Prompt User }
  write('[PgDn], [PgUp] - next, previous screen, [Esc] to exit, Any Other To Sta
  answerSS:=blankSS;
  GETITEM(75,25,1,'C',answerSS,'U','','',retSS,TRUE,7,0);
  fillchar(blank_lineSS[1], 76, ' '); blank_lineSS[0]:=chr(76);
  gotoxy(1,25); write(blank_lineSS);
  if ord(retSS[0])=2 then
  begin
    if (retSS[2]=puSS) or (retSS[2]=pdSS) then
      retSS:=retSS[2]
    else
      retSS:=blankSS;
  end else
    if retSS<>escSS then retSS:=blankSS;
  CASE retSS[1] of
    puSS: use_screenSS:=use_screenSS-1;
    pdSS: use_screenSS:=use_screenSS+1;
    blankSS: begin retSS[0]:=chr(2); retSS[1]:=chr(0); retSS[2]:=uSS end;
    escSS: use_screenSS:=0 ;
  END;
END; { PROCEDURE UPDOWN }

BEGIN { Main Test Program }
  vtypeSS:=SET_MONITOR_TYPE; { 2=Color, 3=Mono }
  screenSS:=0; use_screenSS:=1; BeepOnSS:=false; (* false = no beep *)
  INIT_VAR_1; { B:TITLE }

REPEAT

  CASE use_screenSS of
 1:  begin
```

```
            SCREEN_1; { B:TITLE }
            if retSS = #27                    {if  esc  then exit program}
               then use_screenSS:=0
               else if (retSS[2]=f1SS)        {if  F1  then run SELECT}
                       then
                          begin
                            which_screen := 1;
                            assign(workhorse,'select.chn');
                            chain(workhorse);    {transfer control to select}
                          end;
         end;
      ELSE
         use_screenSS:=0;
      END; { CASE }

    UNTIL use_screenSS=0;

    END. { PROGRAM B:TITLE.PAS }
```

```
(************************* SELECT MENU PROGRAM *********************************)
(*                                                                             *)
(*     This menu program has been generated by a  screen utility program      *)
(*  listed below. Modifications have been made of this program to add global  *)
(*  variable definitions, variables value setting and resetting routines,     *)
(*  input checking codes and additional program flow software.  These         *)
(*  enhancements are made in all the menu programs to different degrees       *)
(*  depending on specific requirements.  This program has been well           *)
(*  documented here as well as in the screen sculptor manuals.                *)
(*                                                                             *)
(*     The function of this menu program is to provide a friendly user        *)
(*  interface for program control and parameter input. when this program is  *)
(*  executed predetermined values are given to these varisbles.  The          *)
(*  operator can change these values and is provided immediate feed back      *)
(*  on the screen.  In this program values presented on the screen takes      *)
(*  precedence over the program header values.  Displayed values can be       *)
(*  updated by simply depressing the carriage return when the cursor is at    *)
(*  the input file name.                                                       *)
(*                                                                             *)
(*                                                           7-1-87           *)
(*******************************************************************************)

PROGRAM SELECT(INPUT,OUTPUT);
{Procedures Generated By Screen Sculptor,Version 1.2 6/22/1987 - 15:51:31 }
CONST CopyrightSS='(C)Copyright 84,85 The Software Bottling Company Of New York
{    DO NOT REMOVE The Above Copyright Notice
     This Program may not be used without the above Copyright Notice
}
{  SCREEN SCULPTOR(C) Version 1.2
   (C) COPYRIGHT, THE SOFTWARE BOTTLING COMPANY OF NEW YORK, 1984, 1985
   WARNING: Do not attempt to make any changes to this procedure unless you
   have made a backup. The Software Bottling Company Of New York can not
   and will not support such changes made to this program.
   Turbo(tm) Pascal Version, Trade Mark Of Borland International}

{ Global Required Definition of variables and constants }

    TYPE                                {writeheader variables}
          id = RECORD
                 nx,                    {nx= # of A-scans in one data set}
                 nz     :integer;       {nx = A-scan lenght, normally 400}
                 w,                     {w = survey width}
                 dt,                    {dt = sampling period}
                 v      :real;          {v = radar propagation velecity}
                 sp     :real;          {sp = averaged data scan-scan spacing}
                 Xavg   :integer;       {Xavg = # of A-scan averaged per data A-scan
                 stn    :integer;       {station = data station}
                 ln     :real;          {ln = data lenght}
                 scns   :integer;       {scns =# of A-scansin the data}
                 mrk    :integer;       {delimiter between A-scans in original data}
                 zoffset:integer;       {template determined sample offset}
                 dcap   :real;          { data capacity of a disk }
                 srcfile:string[40];{scource data file}
                 spare1:real;           {template determined velocity}
```

```
                  spare2:integer;    {spare}
                END;
     VAR    { global variables }
            fname: string[40];         {name of I/O file}
            fin: string[40];           {name of input file}
            workhorse:file;            {chainfile assignee}
            fout:string[40];           {name of output file}
            which_screen:integer;      {which screen to use }
            chain_return:boolean;      {returning from chain program?}
            auto_processing:string[14];{automatic disk processing key}
            {the following characters are used for program flow control}
            char1,                     {plot intensity var in Preview}
            char2,                     {width calibtation var in Preview}
            char3,                     {print var in Preview}
            char4,                     {print var in Transfer}
            char5,                     {disk process var in Transfer}
            char6,                     {multiple processing var in T-saft}
            char7,                     {velocity stepping var in T-saft}
            char8,                     {print image var in T-saft}
            char9,                     {display format var in T-saft}
            char10,                  {spare character variables}
            char11,                  {spare character variables}
            char12,                  {spare character variables}
            char13,                  {spare character variables}
            char14,                  {spare character variables}
            char15:char;             {spare character variables}
            sparename1,
            sparename2,
            sparename3,
            sparename4:string[40];     {spare strings}
            spareI1,
            spareI2,
            spareI3,
            spareI4:integer;           {spare integers}
            spareR1,
            spareR2,
            spareR3,
            spareR4:real;              {spare reals}
            info:id;                   {header information}

{the above are global variables to be passed from progran to program during
 the chain calls. These variables are to be declared exactly as above in all
 programs wether used or not.  Tailoring will be done with the spares.}

{what follows below are translation of the above header variables}

 VAR  points,aslen:real{integer};
      width,tsamp,vel,calspace:real;
      navg,station:real{integer};
      lenght:real;
      scans,mark:real{integer};
      spare:real;
      datcap:real;
      srcfile:string[14];
      detv:real;
      depthoff:real{integer};
      datafilenames1,datafilenames2:string[14];
      datfile:file;
      buff: ARRAY[0..127] OF byte;  {input buffer for blocking data and header}
      OK:boolean;                   {file existence flag}
      indexinc:integer;             {file index increment in multiproc}


TYPE  STR2 = STRING[2]; STR80 = STRING[80]; STR79 = STRING[79];
      resSS = (staySS, prevSS, exitSS, nextSS);

CONST { Esc, Up Arrow Key, Left Arrow Key , Page Up Key        }
      escSS=#27;   uSS='H'; lSS='K'; puSS='I';
      { Blank, Down Arrow Key, Right Arrow Key, Page Down Key }
      blankSS=' '; dSS='P'; rSS='M'; pdSS='Q';
      { Function keys  F1-F10 }
      f1SS=';'; f2SS='<'; f3SS='='; f4SS='>'; f5SS='?';
      f6SS=' '; f7SS='A'; f8SS='B'; f9SS='C'; f10SS='D';
      retSS : STR2='';

VAR   actionSS, last_field_actionSS : resSS;
      hiSS, loSS : REAL;
      vtypeSS, screenSS, use_screenSS, screen_fieldSS, varSS : INTEGER;
      file_existSS, last_fieldSS, retrieveSS : BOOLEAN;
      rangeSS : STR80;
      BeepOnSS: BOOLEAN;


VAR { Variables Section For A:SELECT }
   PREVIEW: STRING[1]; TRANSFER: STRING[1]; TSAFT: STRING[1];
   VELOCITY: STRING[1]; DIR: STRING[1]; REVIEW: STRING[1];
   AUTOPROCESSING: STRING[1];


{$V-,C-,R-} { Pascal Directives. See Compiler Manual }
  {$I TURPROCS.TUR  Include Procedures In This File. See Manual }

(*   This is a summary of the procedures in TURPROCS.TUR

PROCEDURE BEEP(BeepOn: BOOLEAN);            { Sound Beep if BeepOn=TRUE }
PROCEDURE CLEAR_KBD;                        { Clear Keyboard Buffer }
PROCEDURE COLOR(foregr,backgr:BYTE);        { Set Color }
PROCEDURE WRITEC(vtext: STR80);             { Write Chars Using Color }
FUNCTION  SET_MONITOR_TYPE: INTEGER;        { Determine Monitor Type }
{ Display A Screen Sculptor Screen }        { 2=Color, 3=Mono }
PROCEDURE DISPLAY_SCREEN(screen_name: STR80; VAR file_existSS: BOOLEAN);
{ Display And Get An Item From Screen. See Detailed Desription In Manual }
PROCEDURE GETITEM(
                COL,LIN,LEN :          BYTE;     { Column, Line, Length }
                ITYPE :                CHAR;     { Type= C, N, D, Y, M  }
            VAR WITEM :                STR80;    { Variable Name        }
                PICT :                 STR80;    { Picture X, U, L, 9, 8 # }
                ITEM_LOW,ITEM_HIGH :   STR80;    { Range - Numerics/Date Only}
            VAR RET :                    STR2;     { Returned Code        }
                RETRIEVE :               BOOLEAN;  { False=Disp Only, True=Get }
                FGR_COLOR,BGR_COLOR : BYTE         { Colors Foregr, Backgr }
                ); EXTERN;

(*********************** DIRECTORY ******************************************)
(*                                                                           *)
(*   Reads current logged drive unless a mask is specified which            *)
(*   points to another drive or directory.                                   *)
(*   Program ask for mask which is optional. If a mask is specified          *)
(*   then program will look for matching files. If no mask is specified     *)
(*   then default will list all files.  Note, if path is specified then     *)
(*    *.*  must be specified to list all files.                              *)
(*   Further details on this procedure may be obtain from the turbo         *)
(*   Tutor manual by Borland                                                 *)
(****************************************************************************)

Procedure Directory;
{program QDL;}

{-----------------------------------------------------------------------------

    QDL uses MSDos to get a listing of an IBM formated diskette.
  The function calls used can be found in the DOS Technical Reference Manual.
  This program saves the current Data Transfer Area ( DTA ) in the variables
  DTAseg and DTAofs.  The DTA is then reset to the Segment and Offset of
  a Buffer variable 'DTA'.

-----------------------------------------------------------------------------}
  {$I-,U-,C-}


type                                { TYPE declarations }
  Registers =
    record             { register pack used in MSDos call }
      AX, BX, CX, DX, BP, SI, DI, DS, ES, Flags : Integer;
    end;
  Char80arr      = array [ 1..80 ] of Char;
  String80       = string[ 80 ];

var                                  { VARIABLE declarations }
  DTA : array [ 1..43 ] of Byte;       { Data Transfer Area Buffer }
  DTAseg,                              { DTA Segment before exicution }
  DTAofs,                              { DTA Offset                   }
  SetDTAseg,                           { DTA Segment and Offset set after }
  SetDTAofs,                           { start of program }
  Error,                               { Error return }
  I, J,                                { used as counters }
  Option : Integer;                    { used to specify file types }
  Regs : registers;                    { register pack for the DOS call }
  Buffer,                              { generic Buffer }
  NamR : String80;                     { file name }
  Mask : Char80arr;                    { file Mask }
  x,y:integer;                         { x y screen position}
```

```
{---------------------------------------------------------------------------
     SetDTA resets the current DTA to the new address specified in the
parameters 'SEGMENT' and 'OFFSET'.
--------------------------------------------------------------------------}

procedure SetDTA( Segment, Offset : Integer; var Error : Integer );
begin
  Regs.AX := $1A00;          { Function used to set the DTA }
  Regs.DS := Segment;        { store the parameter Segment in DS }
  Regs.DX := Offset;         {                     Offset in DX }
  MSDos( Regs );             { Set DTA location }
  Error := Regs.AX and $FF; { get Error return }
end; { of proc SetDTA }

{---------------------------------------------------------------------------
     GetCurrentDTA is used to get the current Disk Transfer Area ( DTA )
address.  A function code of $2F is stored in the high Byte of the AX
register and a call to the predefined procedure MSDos is made.  This can
also be accomplished by using the  Intr  procedure with the same register
record and a $21 specification for the interrupt.
--------------------------------------------------------------------------}

procedure GetCurrentDTA( var Segment, Offset : Integer;
                         var Error : Integer );
begin
  Regs.AX := $2F00;     { Function used to get current DTA address }
                        { $2F00 is used instead of $2F shl 8 to save
                          three assembly instructions.  An idea for
                          optimization. }
  MSDos( Regs );        { Exicute MSDos function request }
  Segment := Regs.ES;   { Segment of DTA returned by DOS }
  Offset := Regs.BX;    { Offset of DTA returned }
  Error := Regs.AX and $FF;
end; { of proc GetCurrentDTA }


{---------------------------------------------------------------------------
     GetOption returns the code used to find the file names on the current
directory ( ie. hidden, standard, or directory ).
--------------------------------------------------------------------------}

procedure GetOption( var Option : Integer );
begin
  Option := 1;          {standard files}
end; { of proc GetOption }

{---------------------------------------------------------------------------
     GetFirst gets the first directory entry of a particular file Mask.  The
Mask is passed as a parameter 'Mask' and,  the Option was previosly specified
in the SpecifyOption procedure.
--------------------------------------------------------------------------}

procedure GetFirst( Mask : Char80arr; var NamR : String80;
                    Segment, Offset : Integer; Option : Integer;
```

```
                              var Error : Integer );
var
  I : Integer;
begin
  Error := 0;
  Regs.AX := $4E00;            { Get first directory entry }
  Regs.DS := Seg( Mask );      { Point to the file Mask }
  Regs.DX := Ofs( Mask );
  Regs.CX := Option;           { Store the Option }
  MSDos( Regs );               { Exicute MSDos call }
  Error := Regs.AX and $FF;    { Get Error return }
  I := 1;                      { initialize 'I' to the first element }
  repeat                       { Enter the loop that reads in the }
                               { first file entry }
    NamR[ I ] := Chr( mem[ Segment : Offset + 29 + I ] );
    I := I + 1;
  until ( not ( NamR[ I - 1 ] in [ ' '..' ' ] ));
  NamR[ 0 ] := Chr( I - 1 );    { set string length because assigning }
                                { by element does not set length }
end; { of proc GetFirst }

{----------------------------------------------------------------------------
    GetNextEntry uses the first bytes of the DTA for the file Mask, and
returns the next file entry on disk corresponding to the file Mask.
----------------------------------------------------------------------------}

procedure GetNextEntry( var NamR : String80; Segment, Offset : Integer;
                        Option : Integer; var Error : Integer );
var
  I : Integer;
begin
  Error := 0;
  Regs.AX := $4F00;             { Function used to get the next }
                                { directory entry }
  Regs.CX := Option;            { Set the file option }
  MSDos( Regs );                { Call MSDos }
  Error := Regs.AX and $FF;     { get the Error return }
  I := 1;
  repeat
    NamR[ I ] := Chr( mem[ Segment : Offset + 29 + I ] );
    I := I + 1;
  until ( not ( NamR[ I - 1 ] in [ ' '..' ' ] ));
  NamR[ 0 ] := Chr( I - 1 );
end; { of proc GetNextEntry }

{
              main body of program QDL
}

begin
  Writeln(' ********** EXECUTING DIRECTORY *********** ');
  for I := 1 to 21 do DTA[ I ] := 0;  { Initialize the DTA Buffer }
    for I := 1 to 80 do begin         { Initialize the Mask and }
      Mask[ I ] := Chr( 0 );        { file name buffers }
    NamR[ I ] := Chr( 0 );
  end;
NamR[ 0 ] := Chr( 0 );            { Set the file name length to 0 }
WriteLn( 'QDL version 2.00A' );
WriteLn;
GetCurrentDTA( DTAseg, DTAofs, Error );  { Get the current DTA address }
if ( Error <> 0 ) then begin             { Check for errors }
  WriteLn( 'Unable to get current DTA' );
  WriteLn( 'Program aborting.' );         { and abort. }
  Halt;                                   { end program now }
end;
SetDTAseg := Seg( DTA );
SetDTAofs := Ofs( DTA );
SetDTA( SetDTAseg, SetDTAofs, Error );        { Reset DTA addresses }
if ( Error <> 0 ) then begin                  { Check for errors }
  WriteLn( 'Cannot reset DTA' );          { Error message }
  WriteLn( 'Program aborting.' );
  Halt;                                   { end program }
end;
Error := 0;
Buffer[ 0 ] := Chr( 0 );                      { Set Buffer length to 0 }
GetOption( Option );                          { Get file Option }
clrscr;
gotoxy(5,20);
if ( Option <> 8 ) then begin
  Write( 'File Mask : ' );                { prompt }
  ReadLn( Buffer );
  WriteLn;
end;
if ( length( Buffer ) = 0 ) then              { if nothing was entered }
  Buffer := '????????.???';                  { then use global search }
for I := 1 to length( Buffer ) do        { Assign Buffer to Mask }
  Mask[ I ] := Buffer[ I ];
GetFirst( Mask, NamR, SetDTAseg, SetDTAofs, Option, Error );
if ( Error = 0 ) then begin             { Get the first directory entry }
  if ( Option <> 8 ) then begin         { if not volume label }
    WriteLn( 'Directory of : ', Buffer ); { Write directory message }
    WriteLn;
  end;
  Write( NamR )
end
else if ( Option = 8 ) then
  WriteLn( 'Volume label not found.' )
else WriteLn( 'File ''', Buffer, ''' not found.' );
while ( Error = 0 ) do begin
  GetNextEntry( NamR, SetDTAseg, SetDTAofs, Option, Error );
  if ( Error = 0 ) then begin  {list file names in five vertical columns}
                          x := whereX;
                          y := whereY;
                          x := (x div 15) * 15 + 15;
                          if x > 61 then
                                       begin
                                          x := 1;
                                          y := y + 1;
```

```
                                               if y >25 then begin
                                                            y := 25;
                                                            writeln;
                                                           end;
                                             end;
                                  gotoxy(x,y);
                                  Write( NamR );
                                end;
          end;
          SetDTA( DTAseg, DTAofs, Error );
        end; { procedure directory }


        PROCEDURE RET_STATUS;
        { Check Status Of Variable retSS and return a code in 'actionSS'   set 'varSS'
          This procedure is called immediately following GETITEM }

        { Input to this procedure:
          when retSS is length 1 the values are any of the ASCII chars
          when retSS is length 2 the values are uSS, lSS, puSS, pdSS, function keys
                                              dSS, rSS
                                              ( See CONST Section For Meanings ) }
        { Output:
          The following codes are returned in actionSS : nextSS, prevSS,
                                                     exitSS, staySS }
        { Based upon 'actionSS' this procedure will then set 'varSS' to an integer,
          which represents the next item (variable ) to get.  }
        BEGIN
          last_field_actionSS:=staySS;
          actionSS:=nextSS; { Initialize Action Code }
          if retrieveSS then { Is retrieveSS TRUE? }
          begin
            if ord(retSS[0])=2 then { Is retSS length 2 ? }
            begin
              CASE retSS[2] of
              { Action to be taken depending on the last key pressed }
                uSS, lSS: actionSS:=prevSS;  { Up Key, Left Key }
                dSS, rSS: actionSS:=nextSS;  {Down Key, Right Key}
                puSS: actionSS:= { Page Up   }staySS;
                pdSS: actionSS:= { Page Down }staySS;
                f1SS: actionSS:= {execute} exitSS;          (**** exits screen ****)
                f2SS,f3SS,f4SS,f5SS,
                f6SS,f7SS,f8SS,f9SS,f10SS: actionSS:=staySS; { Function Key }
              END { Case ret };
            end else { retSS is length 1 }
            begin
              if retSS=escSS { Escape Key } then actionSS:=exitSS
            end;
          { Any other key not in the above list will keep actionSS=nextSS }
          end; {retrieveSS}
          CASE actionSS of
            staySS: ;
            nextSS: begin
                      varSS:=varSS+1; if varSS>screen_fieldSS then varSS:=1;
              if last_fieldSS and retrieveSS then actionSS:=last_field_actionSS
            end;
    prevSS: begin varSS:=varSS-1; if varSS<1 then varSS:=screen_fieldSS end;
    exitSS: ;
  END; { CASE }
END; {PROCEDURE RET_STATUS}

PROCEDURE GETNUM(
                COL,LIN,LEN :          BYTE;      { Column, Line, Length }
                ITYPE :                CHAR;      { Type= C, N, D, Y, M  }
            VAR WITEM :                REAL;      { Numerci Variable Name }
                PICT :                 STR80;     { Picture X, U, L, 9, 8 # }
                ITEM_LOW,ITEM_HIGH :   REAL;      { Range - Numerics/Date Only}
            VAR RET :                  STR2;      { Returned Code          }
                RETRIEVE :             BOOLEAN;   { False=Disp Only, True=Get }
                FGR_COLOR,BGR_COLOR :  BYTE       { Colors Foregr, Backgr }
                );
{ This Procedure converts numeric to string before calling GETITEM }
{ It then converts the result back to numeric }
VAR numSS,numloSS,numhiSS: STR80; errorcodeSS,dec_posSS: INTEGER;
BEGIN
  { Get # of Decimal Positions }
  dec_posSS:=ord(pict[0])-pos('.',pict);
  { Convert item, low and high range to string }
  STR(witem:0:dec_posSS,numSS);
  STR(item_low:0:dec_posSS,numloSS);
  STR(item_high:0:dec_posSS,numhiSS);
  GETITEM(col,lin,len,itype,numSS,pict,numloSS,numhiSS,
          ret,retrieve,fgr_color,bgr_color);
  { Convert string to numeric item }
  VAL(numSS, witem, errorcodeSS);
END; { GETNUM }


(*VAR { Variables Section For A:SELECT } {moved to declaration area}
   PREVIEW: STRING[1]; TRANSFER: STRING[1]; TSAFT: STRING[1];
   VELOCITY: STRING[1]; DIRECTORY: STRING[1]; REVIEW: STRING[1];
   AUTOPROCESSING: STRING[1];                                     *)

PROCEDURE INIT_VAR_1; { B:SELECT }
{ Initialize Variables To Default Values }
BEGIN
    PREVIEW:=''; TRANSFER:=''; TSAFT:=''; VELOCITY:=''; DIR:='';
    REVIEW:=''; AUTOPROCESSING:='';
END; { PROCEDURE INIT_VARS_1}

PROCEDURE SCREEN_1; { B:SELECT }
BEGIN

if screenSS<>1 then     {display this screen if different from present screen}
begin
  screenSS:=1; screen_fieldSS:=7; varSS:=1;
  retrieveSS:=FALSE; last_fieldSS:=FALSE;
  DISPLAY_SCREEN('SELECT.SCR',file_existSS);  { Display Screen }
```

```
    if not file_existSS then
    begin gotoxy(1,1); write('Missing Screen SELECT') end;
  end;
  retSS:='';

  { Display Items. Change retrieveSS to TRUE and INPUT items}
  REPEAT { until actionSS = exitSS }
    CASE varSS of
  1:  GETITEM(7,19,1,'C',PREVIEW,       {displays and prompt for menu entries}
       'X','','',retSS,retrieveSS,14,0);
  2:  GETITEM(24,19,1,'C',TRANSFER,
       'X','','',retSS,retrieveSS,14,0);
  3:  GETITEM(41,19,1,'C',TSAFT,
       'X','','',retSS,retrieveSS,14,0);
  4:  GETITEM(58,19,1,'C',VELOCITY,
       'X','','',retSS,retrieveSS,14,0);
  5:  GETITEM(7,21,1,'C',DIR,
       'X','','',retSS,retrieveSS,14,0);
  6:  GETITEM(24,21,1,'C',REVIEW,
       'X','','',retSS,retrieveSS,14,0);
  7:  GETITEM(41,21,1,'C',AUTOPROCESSING,
       'X','','',retSS,retrieveSS,14,0);
    END; { CASE }

    if varSS=screen_fieldSS then last_fieldSS:=TRUE;
    RET_STATUS; { Check the code in  retSS . Set   varSS  and  actionSS  }

  { Check to see whether to switch retrieveSS to true }
  if last_fieldSS and (not retrieveSS) then
  begin
    retrieveSS:=TRUE; last_fieldSS:=FALSE; actionSS:=staySS; varSS:=1;
  end else
    last_fieldSS:=FALSE;

  UNTIL actionSS=exitSS
  END; { PROCEDURE SCREEN_1 }


  BEGIN { Main Test Program }
    vtypeSS:=SET_MONITOR_TYPE; { 2=Color, 3=Mono }
    screenSS:=0; use_screenSS:=1; BeepOnSS:=false;      {no beep}
    INIT_VAR_1; { B:SELECT }
    chain_return := false;
    auto_processing := '';             {reset to no automatic processing}

  REPEAT

    CASE use_screenSS of
     1:  begin
           SCREEN_1; { B:SELECT }
             if retSS = #27 then         { esc  returns to title screen}
                         begin
                           assign(workhorse,'NCEL.com');
                           execute(workhorse);
```

```
                   end
               else if retSS[2] = f1SS then
               begin
                case varSS of

                  1: begin              {execute data Preview}
                       assign(workhorse,'Preview.chn');
                       chain(workhorse);
                     end;

                  2: begin              {execute data Transfer}
                       assign(workhorse,'Transfer.chn');
                       chain(workhorse);
                     end;

                  3: begin              {execute T-saft imaging}
                       assign(workhorse,'Tsaft.chn');
                       chain(workhorse);
                     end;

                  4: begin              {execute Velocity determination}
                       assign(workhorse,'Velocity.chn');
                       chain(workhorse);
                     end;

                  5: begin              {execute directory displaying}
                       Directory;
                       screenSS := 0;
                       writeln;
                       write('  Press Enter to return to menu  ');
                       readln;
                     end;

                  6: begin              {execute file Review}
                       assign(workhorse,'Review.chn');
                       chain(workhorse);
                     end;
                  7: begin              {set and start automatic processing}
                       auto_processing := 'Autoprocess';
                       assign(workhorse,'Preview.chn');
                       chain(workhorse);
                     end;
                end;
               end;
           end;

      ELSE
         use_screenSS:=0;            {set for redisplay of screen}
      END; { CASE }

    UNTIL use_screenSS=0;            {redisplay of original screen}

    END. { PROGRAM B:SELECT.PAS }
```

```
(****************************** PREVIEW MENU PROGRAM ****************************)
(*                                                                              *)
(*     This menu program has been generated by a  screen utility program        *)
(*  listed below. Modifications have been made of this program to add global    *)
(*  variable definitions, variables value setting and resetting routines,       *)
(*  input checking codes and additional program flow software.  These           *)
(*  enhancements are made in all the menu programs to different degrees         *)
(*  depending on specific requirements.  This program has been well             *)
(*  documented here as well as in the screen sculptor manuals.                  *)
(*                                                                              *)
(*     The function of this menu program is to provide a friendly user          *)
(*  interface for program control and parameter input. when this program is     *)
(*  executed predetermined values are given to these varisbles.  The            *)
(*  operator can change these values and is provided immediate feed back        *)
(*  on the screen.  In this program values presented on the screen takes        *)
(*  precedence over the program header values.  Displayed values can be         *)
(*  updated by simply depressing the carriage return when the cursor is at      *)
(*  the input file name.                                                        *)
(*                                                                              *)
(*                                                          7-1-87              *)
(********************************************************************************)

PROGRAM PREVIEW(input,output);
{Procedures Generated By Screen Sculptor,Version 1.2 5/19/1987 - 13:53:32 }
CONST CopyrightSS='(C)Copyright 84,85 The Software Bottling Company Of New York';
{    DO NOT REMOVE The Above Copyright Notice
     This Program may not be used without the above Copyright Notice
}
{  SCREEN SCULPTOR(C) Version 1.2
   (C) COPYRIGHT, THE SOFTWARE BOTTLING COMPANY OF NEW YORK, 1984, 1985
   WARNING: Do not attempt to make any changes to this procedure unless you
   have made a backup. The Software Bottling Company Of New York can not
   and will not support such changes made to this program.
   Turbo(tm) Pascal Version, Trade Mark Of Borland International}

{ Global Required Definition of variables and constants }

    TYPE                               {writeheader variables}
          id = RECORD
                 nx,                   {nx= # of A-scans in one data set}
                 nz    :integer;       {nx = A-scan lenght, normally 400}
                 w,                    {w = survey width}
                 dt,                   {dt = sampling period}
                 v     :real;          {v = radar propagation velecity}
                 sp    :real;          {sp = averaged data scan-scan spacing}
                 Xavg  :integer;       {Xavg = # of A-scan averaged per data A-scan}
                 stn   :integer;       {station = data station}
                 ln    :real;          {ln = data lenght}
                 scns  :integer;       {scns =# of A-scansin the data}
                 mrk   :integer;       {delimiter between A-scans in original data}
                 zoffset:integer;      {template determined sample offset}
                 dcap  :real;          { data capacity of a disk }
                 srcfile:string[40];{scource data file}
                 spare1:real;          {template determined velocity}
```

```
                spare2:integer;     {spare}
              END;
      VAR   { global variables }
            fname: string[40];        {name of I/O file}
            fin: string[40];          {name of input file}
            workhorse:file;           {chainfile assignee}
            fout:string[40];          {name of output file}
            which_screen:integer;     {which screen to use }
            chain_return:boolean;     {returning from chain program?}
            auto_processing:string[14];{automatic disk processing key}
            {the following characters are used for program flow control}
            char1,                    {plot intensity var in Preview}
            char2,                    {width calibtation var in Preview}
            char3,                    {print var in Preview}
            char4,                    {print var in Transfer}
            char5,                    {disk process var in Transfer}
            char6,                    {multiple processing var in T-saft}
            char7,                    {velocity stepping var in T-saft}
            char8,                    {print image var in T-saft}
            char9,                    {display format var in T-saft}
            char10,                {spare character variables}
            char11,                {spare character variables}
            char12,                {spare character variables}
            char13,                {spare character variables}
            char14,                {spare character variables}
            char15:char;           {spare character variables}
            sparename1,
            sparename2,
            sparename3,
            sparename4:string[40];    {spare strings}
            spareI1,
            spareI2,
            spareI3,
            spare14:integer;          {spare integers}
            spareR1,
            spareR2,
            spareR3,
            spareR4:real;             {spare reals}
            info:id;                  {header information}

{the above are global variables to be passed from progran to program during
 the chain calls. These variables are to be declared exactly as above in all
 programs wether used or not.  Tailoring will be done with the spares.}


{what follows below are translation of the above header variables}

 VAR  points,aslen:real{integer};
      width,tsamp,vel,calspace:real;
      navg,station:real{integer};
      lenght:real;
      scans,marker:real{integer};
      spare:real;
      datcap:real;
```

```
     srcfile:string[40];
     detv:real;
     depthoff:real{integer};
     datfile:file;
     OK:boolean;

TYPE STR2 = STRING[2]; STR80 = STRING[80]; STR79 = STRING[79];
     resSS = (staySS, prevSS, exitSS, nextSS);

CONST { Esc, Up Arrow Key, Left Arrow Key , Page Up Key        }
     escSS=#27;   uSS='H'; lSS='K'; puSS='I';
     { Blank, Down Arrow Key, Right Arrow Key, Page Down Key }
     blankSS=' '; dSS='P'; rSS='M'; pdSS='Q';
     { Function keys  F1-F10 }
     f1SS=';'; f2SS='<'; f3SS='='; f4SS='>'; f5SS='?';
     f6SS=' '; f7SS='A'; f8SS='B'; f9SS='C'; f10SS='D';
     retSS : STR2='';

VAR  actionSS, last_field_actionSS : resSS;
     hiSS, loSS : REAL;
     vtypeSS, screenSS, use_screenSS, screen_fieldSS, varSS : INTEGER;
     file_existSS, last_fieldSS, retrieveSS : BOOLEAN;
     rangeSS : STR80;
     BeepOnSS: BOOLEAN;

VAR { Variables Section For A:PREVIEW/AUTOPROC }
   OPERATOR: STRING[20];{ FIN: STRING[40]; FNAME: STRING[40]; VEL: REAL;
   DEPTHOFF: REAL; STATION: REAL; WIDTH: REAL; DATCAP: REAL; TSAMP: REAL;
   MARKER: REAL; CALSPACE: REAL; POINTS: REAL; ASLEN: REAL;}
   INTENSITY: STRING[1]; CALIBRATE: STRING[1]; PRINTPRV: STRING[1];
   PRINTTRANS: STRING[1]; PRINTSAFT: STRING[1]; VELSTEPPING: STRING[1];



  {$V-,C-,R-} { Pascal Directives. See Compiler Manual }
  {$I TURPROCS.TUR  Include Procedures In This File. See Manual }

(*   This is a summary of the procedures in TURPROCS.TUR

PROCEDURE BEEP(BeepOn: BOOLEAN);            { Sound Beep if BeepOn=TRUE }
PROCEDURE CLEAR_KBD;                        { Clear Keyboard Buffer }
PROCEDURE COLOR(foregr,backgr:BYTE);        { Set Color }
PROCEDURE WRITEC(vtext: STR80);             { Write Chars Using Color }
FUNCTION  SET_MONITOR_TYPE: INTEGER;        { Determine Monitor Type }
{ Display A Screen Sculptor Screen }        { 2=Color, 3=Mono }
PROCEDURE DISPLAY_SCREEN(screen_name: STR80; VAR file_existSS: BOOLEAN);
{ Display And Get An Item From Screen. See Detailed Desription In Manual }
PROCEDURE GETITEM(
                COL,LIN,LEN :         BYTE;     { Column, Line, Length }
                ITYPE :               CHAR;     { Type= C, N, D, Y, M  }
            VAR WITEM :               STR80;    { Variable Name        }
                PICT :                STR80;    { Picture X, U, L, 9, 8 # }
                ITEM_LOW,ITEM_HIGH :  STR80;    { Range - Numerics/Date Only}
```

```
              VAR RET :                 STR2;     { Returned Code         }
                  RETRIEVE :            BOOLEAN;  { False=Disp Only, True=Get }
                  FGR_COLOR,BGR_COLOR : BYTE      { Colors Foregr, Backgr }
                  ); EXTERN;

*)

PROCEDURE RET_STATUS;
{ Check Status Of Variable retSS and return a code in 'actionSS'   set 'varSS'
  This procedure is called immediately following GETITEM }

{ Input to this procedure:
  when retSS is length 1 the values are any of the ASCII chars
  when retSS is length 2 the values are uSS, lSS, puSS, pdSS, function keys
                                        dSS, rSS
                                        ( See CONST Section For Meanings ) }
{ Output:
  The following codes are returned in actionSS : nextSS, prevSS,
                                                 exitSS, staySS }
{ Based upon 'actionSS' this procedure will then set 'varSS' to an integer,
  which represents the next item (variable ) to get.  }
BEGIN
  last_field_actionSS:=staySS;
  actionSS:=nextSS; { Initialize Action Code }
  if retrieveSS then { Is retrieveSS TRUE? }
  begin
    if ord(retSS[0])=2 then { Is retSS length 2 ? }
    begin
      CASE retSS[2] of
      { Action to be taken depending on the last key pressed }
        uSS, lSS: actionSS:=prevSS;  { Up Key, Left Key }
        dSS, rSS: actionSS:=nextSS;  {Down Key, Right Key}
        puSS: actionSS:= { Page Up   }staySS;
        pdSS: actionSS:= { Page Down }staySS;
        f1SS: actionSS:= {execute} exitSS;          (**** exits screen ****)
        f2SS,f3SS,f4SS,f5SS,
        f6SS,f7SS,f8SS,f9SS,f10SS: actionSS:=staySS; { Function Key }
      END { Case ret };
    end else { retSS is length 1 }
    begin
      if retSS=escSS { Escape Key } then actionSS:=exitSS
    end;
  { Any other key not in the above list will keep actionSS=nextSS }
  end; {retrieveSS}
  CASE actionSS of
    staySS: ;
    nextSS: begin
              varSS:=varSS+1; if varSS>screen_fieldSS then varSS:=1;
              if last_fieldSS and retrieveSS then actionSS:=last_field_actionSS
            end;
    prevSS: begin varSS:=varSS-1; if varSS<1 then varSS:=screen_fieldSS end;
    exitSS: ;
  END; { CASE }
END; {PROCEDURE RET_STATUS}
```

```
PROCEDURE GETNUM(
                COL,LIN,LEN :          BYTE;     { Column, Line, Length }
                ITYPE :                CHAR;     { Type= C, N, D, Y, M  }
            VAR WITEM :                REAL;     { Numerci Variable Name }
                PICT :                 STR80;    { Picture X, U, L, 9, 8 # }
                ITEM_LOW,ITEM_HIGH :   REAL;     { Range - Numerics/Date Only}
            VAR RET :                  STR2;     { Returned Code         }
                RETRIEVE :             BOOLEAN;  { False=Disp Only, True=Get }
                FGR_COLOR,BGR_COLOR : BYTE       { Colors Foregr, Backgr }
                );
{ This Procedure converts numeric to string before calling GETITEM }
{ It then converts the result back to numeric }
VAR numSS,numloSS,numhiSS: STR80; errorcodeSS,dec_posSS: INTEGER;
BEGIN
  { Get # of Decimal Positions }
  dec_posSS:=ord(pict[0])-pos('.',pict);
  { Convert item, low and high range to string }
  STR(witem:0:dec_posSS,numSS);
  STR(item_low:0:dec_posSS,numloSS);
  STR(item_high:0:dec_posSS,numhiSS);
  GETITEM(col,lin,len,itype,numSS,pict,numloSS,numhiSS,
          ret,retrieve,fgr_color,bgr_color);
  { Convert string to numeric item }
  VAL(numSS, witem, errorcodeSS);
END; { GETNUM }



(*VAR { Variables Section For A:PREVIEW/AUTOPROC }{moved to declaration area}
   OPERATOR: STRING[20]; FIN: STRING[40]; FNAME: STRING[40]; VEL: REAL;
   DEPTHOFF: REAL; STATION: REAL; WIDTH: REAL; DATCAP: REAL; TSAMP: REAL;
   MARKER: REAL; CALSPACE: REAL; POINTS: REAL; ASLEN: REAL;
   INTENSITY: STRING[1]; CALIBRATE: STRING[1]; PRINTPRV: STRING[1];
   PRINTTRANS: STRING[1]; PRINTSAFT: STRING[1]; VELSTEPPING: STRING[1];*)

PROCEDURE INIT_VAR_1; { A:PREVIEW/AUTOPROC }
{ Initialize Variables To Default Values }
BEGIN
    OPERATOR:='Mel'; FIN:='c: Data Demo.dat'; FNAME:='c: Data Demo';
    VEL:=0.4000; DEPTHOFF:=0; STATION:=0; WIDTH:=30.000; DATCAP:=0;
    TSAMP:=0.1600; MARKER:=0; CALSPACE:=0.32258; POINTS:=32; ASLEN:=400;
    INTENSITY:='L'; CALIBRATE:='Y'; PRINTPRV:='N'; PRINTTRANS:='N';
    PRINTSAFT:='N'; VELSTEPPING:='N';
END; { PROCEDURE INIT_VARS_1}

(****** combined screen input output for auto processing and preview  ********)
(*   the screen displayed is dependent on selection at the selection menu   *)

PROCEDURE SCREEN_1; { A:PREVIEW/AUTOPROC }
BEGIN

if screenSS<>1 then           {redisplay screen only if present screen <> 1}
begin
  screenSS:=1; varSS:=1;
  retrieveSS:=FALSE; last_fieldSS:=FALSE;
  clrscr;
  if auto_processing = 'Autoprocess'      {auto processing screen selection}
    then                                  {display Auto Process menu}
      begin
        screen_fieldSS:=19;      {nineteen parameters in auto processing screen
        DISPLAY_SCREEN('AUTOPROC.SCR',file_existSS);  { Display Screen }
        if not file_existSS then
           begin gotoxy(1,1); write('Missing Screen AUTOPROC') end;
      end
    else
      begin                    {display Preview menu}
       screen_fieldSS:= 16;     {only sixteen parameters in preview screen}
       DISPLAY_SCREEN('PREVIEW.SCR',file_existSS);  { Display Screen }
       if not file_existSS then
          begin gotoxy(1,1); write('Missing Screen PREVIEW') end;
     end;
end;
retSS:='';

{ Display Items. Change retrieveSS to TRUE and INPUT items}
REPEAT { until actionSS = exitSS }
  CASE varSS of
1:  GETITEM(54,12,20,'C',OPERATOR,
     'XXXXXXXXXXXXXXXXXXXX','','',retSS,retrieveSS,15,0);
2:  GETITEM(24,13,40,'C',FIN,
     'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX','','',retSS,retrieveSS,15,0);
3:  GETITEM(24,14,40,'C',FNAME,
     'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX','','',retSS,retrieveSS,15,0);
4:  GETNUM(23,15,7,'N',VEL,
     '##.####',0.0000,1.0000,retSS,retrieveSS,15,0);
5:  GETNUM(23,16,7,'N',DEPTHOFF,
     '#######',-1024.0,1024.0,retSS,retrieveSS,15,0);
6:  GETNUM(23,17,7,'N',STATION,
     '#######',-999999.0,9999999.0,retSS,retrieveSS,15,0);
7:  GETNUM(23,18,7,'N',WIDTH,
     '###.###',0.000,999.999,retSS,retrieveSS,15,0);
8:  GETNUM(23,19,7,'N',DATCAP,
     '#######',0.0,9999999.0,retSS,retrieveSS,15,0);
9:  GETNUM(23,20,7,'N',TSAMP,
     '##.####',0.0000,9.9999,retSS,retrieveSS,15,0);
10:  GETNUM(23,21,7,'N',MARKER,
     '#######',0.0,255.0,retSS,retrieveSS,15,0);
11:  GETNUM(23,22,7,'N',CALSPACE,
     '#.#####',0.00000,9.99999,retSS,retrieveSS,15,0);
12:  GETNUM(23,23,7,'N',POINTS,
     '#######',0.0,32.0,retSS,retrieveSS,15,0);
13:  GETNUM(61,15,7,'N',ASLEN,
     '#######',0.0,512.0,retSS,retrieveSS,15,0);
14:  GETITEM(70,16,1,'C',INTENSITY,
     'U','','',retSS,retrieveSS,15,0);
15:  GETITEM(70,17,1,'Y',CALIBRATE,
     'U','','',retSS,retrieveSS,15,0);
16:  GETITEM(70,18,1,'Y',PRINTPRV,
     'U','','',retSS,retrieveSS,15,0);
17:  GETITEM(70,19,1,'Y',PRINTTRANS,
     'U','','',retSS,retrieveSS,15,0);
18:  GETITEM(70,20,1,'Y',PRINTSAFT,
     'U','','',retSS,retrieveSS,15,0);
19:  GETITEM(70,21,1,'Y',VELSTEPPING,
     'U','','',retSS,retrieveSS,15,0);
  END; { CASE }

  if varSS=screen_fieldSS then last_fieldSS:=TRUE;
  RET_STATUS; { Check the code in  retSS . Set   varSS  and  actionSS  }


{ Check to see whether to switch retrieveSS to true }
if last_fieldSS and (not retrieveSS) then
begin
  retrieveSS:=TRUE; last_fieldSS:=FALSE; actionSS:=staySS; varSS:=1;
end else
  last_fieldSS:=FALSE;

UNTIL actionSS=exitSS
END; { PROCEDURE SCREEN_1 }


(*************************** SET HEADER ****************************************)
(*                                                                              *)
(*     set global header parameters to the local input parameters values        *)
(*                                                                              *)
(*******************************************************************************)

PROCEDURE SET_HEADER;
begin
     with info do                       {set header values}
          begin
                 nx      := trunc(points);
                 nz      := trunc(aslen);
                 w       := width;
                 dt      := tsamp*1.0e-9;
                 v       := vel*1.0e9;
                 sp      := calspace;
                { Xavg    := trunc(navg);}
                 stn     := trunc(station);
                { ln      := lenght;        }
                { scns    := trunc(scans); }
                 mrk     := trunc(marker);
                 dcap    := datcap;
                 srcfile :=srcfile;
                 zoffset := trunc(depthoff);
           END;
       char1 := intensity;
       char2 := calibrate;
       char3 := printprv;
       char4 := printtrans;
       char7 := velstepping;
       char8 := printsaft;
       if auto_processing = 'Autoprocess' then
         begin  {these variables set here will not change during auto process}
           char5 := 'Y';          {set auto disk processing in Transfer}
           char6 := 'Y';          {set multifile processing in T-saft}
           char9 := 'W';          {set wiggle display format in T-saft}
         end;
end; {set header variables}


(***************************** SET VAR ***************************************)
(*                                                                            *)
(*  set program local variables to the global header parameter values        *)
(*                                                                            *)
(******************************************************************************)

PROCEDURE SET_VAR;
    BEGIN
        with info do                   {set header values}
          begin
                 points := nx;
                 aslen := nz;
                 width := w;
                 tsamp := dt*1.0e9;
                 vel := v*1.0e-9;
                 calspace := sp;
                 navg := Xavg;
                 station := stn;
                 lenght := ln;
                 scans := scns;
                 marker := mrk;
                 datcap := dcap;
                 fin := srcfile;
                 depthoff := zoffset;
      end;
 END; {SET_VAR}


BEGIN { Main Test Program }
  vtypeSS:=SET_MONITOR_TYPE; { 2=Color, 3=Mono }
  screenSS:=0; use_screenSS:=1; BeepOnSS:=false;  (*** no beep ***)
  station := 0;            {initialize station}

  INIT_VAR_1; { preview }
if chain_return then set_var;  {set menu entries to previous values}
 gotoxy(5,25);write('datcap=',datcap);
REPEAT

  CASE use_screenSS of
 1:  begin  {case1}
        SCREEN_1; { C:PREVIEW }
        if retSS = #27
            then begin {exit}
                   chain_return := false;
```

```
                  assign(workhorse,'select.chn'); {if  esc  return to select}
                  chain(workhorse);          {transfer control to select}
                end   {exit}
             else if (retSS[2]=f1SS)  then   {if  F1  then execute workhorse}
                begin    {F1}
                 while pos(' ',fin)=1 do  {remove leading blanks}
                 delete(fin,1,1);
                 while (pos(' ',fin)<>0) and (pos(' ',fin)<pos('.',fin))
                   do delete(fin,pos(' ',fin),1);  {delete internal blanks}
                 assign(datfile,fin);
                 {$I-} reset(datfile) {$I+};
                 OK := (IOresult = 0);
                 if not OK                  {unsuccessful open}
               then  begin   {not ok}
                      gotoxy(2,25);
                      write('Cannot find file ',fin,'                    ');
                      delay(2000);           {clear error message}
                      gotoxy(1,25);
                      write('                                               ');
                   end     {not ok}
               else
                 begin   {ok}
                   set_header;
                   clrscr;
                       begin
                         assign(workhorse,'previewk.chn');
                         chain(workhorse);
                       end;
                 end;  {ok}
                end;   {F1}
     end;  {case1}
  ELSE
     use_screenSS:=0;
  END; { CASE }
UNTIL use_screenSS=0;

END. { PROGRAM AUTOPROC/PREVIEW }
```

```
(************************* TRANSFER MENU PROGRAM ******************************)
(*                                                                            *)
(*     This menu program has been generated by a  screen utility program     *)
(*  listed below. Modifications have been made of this program to add global *)
(*  variable definitions, variables value setting and resetting routines,    *)
(*  input checking codes and additional program flow software.  These        *)
(*  enhancements are made in all the menu programs to different degrees      *)
(*  depending on specific requirements.  This program has been well          *)
(*  documented here as well as in the screen sculptor manuals.               *)
(*                                                                            *)
(*     The function of this menu program is to provide a friendly user       *)
(*  interface for program control and parameter input. when this program is  *)
(*  executed predetermined values are given to these varisbles.  The         *)
(*  operator can change these values and is provided immediate feed back     *)
(*  on the screen.  In this program values presented on the screen takes     *)
(*  precedence over the program header values.  Displayed values can be      *)
(*  updated by simply depressing the carriage return when the cursor is at   *)
(*  the input file name.                                                     *)
(*                                                                            *)
(*                                                          7-1-87           *)
(******************************************************************************)

PROGRAM TRANSFER(INPUT,OUTPUT);
{Procedures Generated By Screen Sculptor,Version 1.2 6/10/1987 - 15:18:14 }
CONST CopyrightSS='(C)Copyright 84,85 The Software Bottling Company Of New York';
{    DO NOT REMOVE The Above Copyright Notice
     This Program may not be used without the above Copyright Notice
}
{  SCREEN SCULPTOR(C) Version 1.2
   (C) COPYRIGHT, THE SOFTWARE BOTTLING COMPANY OF NEW YORK, 1984, 1985
   WARNING: Do not attempt to make any changes to this procedure unless you
   have made a backup. The Software Bottling Company Of New York can not
   and will not support such changes made to this program.
   Turbo(tm) Pascal Version, Trade Mark Of Borland International}

{ Global Required Definition of variables and constants }


    TYPE                            {writeheader variables}
          id = RECORD
                 nx,                {nx= # of A-scans in one data set}
                 nz     :integer;   {nx = A-scan lenght, normally 400}
                 w,                 {w = survey width}
                 dt,                {dt = sampling period}
                 v      :real;      {v = radar propagation velecity}
                 sp     :real;      {sp = averaged data scan-scan spacing}
                 Xavg   :integer;   {Xavg = # of A-scan averaged per data A-scan}
                 stn    :integer;   {station = data station}
                 ln     :real;      {ln = data lenght}
                 scns   :integer;   {scns =# of A-scansin the data}
                 mrk    :integer;   {delimiter between A-scans in original data}
                 zoffset:integer;   {template determined sample offset}
                 dcap   :real;      { data capacity of a disk }
                 srcfile:string[40];{scource data file}
```

```
                spare1:real;      {template determined velocity}
                spare2:integer;   {spare}
              END;
     VAR    { global variables }
            fname: string[40];          {name of I/O file}
            fin: string[40];            {name of input file}
            workhorse:file;             {chainfile assignee}
            fout:string[40];            {name of output file}
            which_screen:integer;       {which screen to use }
            chain_return:boolean;       {returning from chain program?}
            auto_processing:string[14];{automatic disk processing key}
            {the following characters are used for program flow control}
            char1,                      {plot intensity var in Preview}
            char2,                      {width calibtation var in Preview}
            char3,                      {print var in Preview}
            char4,                      {print var in Transfer}
            char5,                      {disk process var in Transfer}
            char6,                      {multiple processing var in T-saft}
            char7,                      {velocity stepping var in T-saft}
            char8,                      {print image var in T-saft}
            char9,                      {display format var in T-saft}
            char10,                 {spare character variables}
            char11,                 {spare character variables}
            char12,                 {spare character variables}
            char13,                 {spare character variables}
            char14,                 {spare character variables}
            char15:char;            {spare character variables}
            sparename1,
            sparename2,
            sparename3,
            sparename4:string[40];      {spare strings}
            spareI1,
            spareI2,
            spareI3,
            spare14:integer;            {spare integers}
            spareR1,
            spareR2,
            spareR3,
            spareR4:real;               {spare reals}
            info:id;                    {header information}

{the above are global variables to be passed from progran to program during
 the chain calls. These variables are to be declared exactly as above in all
 programs wether used or not.  Tailoring will be done with the spares.}


{what follows below are translation of the above header variables}

 VAR  points,aslen:real{integer};
      width,tsamp,vel,calspace:real;
      navg,station:real{integer};
      lenght:real;
      scans,mark:real{integer};
      datcap:real;
      srcfile:string[14];
      depthoff:real{integer};
      datfile:file;
      buff: ARRAY[0..127] OF byte;  {input buffer for blocking data and header}
      OK:boolean;                   {file existence flag}
      giveup:boolean;               {flag for giving up looking for data file}

TYPE  STR2 = STRING[2]; STR80 = STRING[80]; STR79 = STRING[79];
      resSS = (staySS, prevSS, exitSS, nextSS);

CONST { Esc, Up Arrow Key, Left Arrow Key , Page Up Key        }
      escSS=#27;   uSS='H'; lSS='K'; puSS='I';
      { Blank, Down Arrow Key, Right Arrow Key, Page Down Key }
      blankSS=' '; dSS='P'; rSS='M'; pdSS='Q';
      { Function keys  F1-F10 }
      f1SS=';'; f2SS='<'; f3SS='='; f4SS='>'; f5SS='?';
      f6SS=' '; f7SS='A'; f8SS='B'; f9SS='C'; f10SS='D';
      retSS : STR2='';

VAR   actionSS, last_field_actionSS : resSS;
      hiSS, loSS : REAL;
      vtypeSS, screenSS, use_screenSS, screen_fieldSS, varSS: INTEGER;
      file_existSS, last_fieldSS, retrieveSS : BOOLEAN;
      rangeSS : STR80;
      BeepOnSS: BOOLEAN;

VAR { Variables Section For B:TRANSFER }
   OPERATOR: STRING[20]; {Fname: STRING[40]; FOUT: STRING[40];}
   PRINTTRANS: STRING[1]; PROCESS: STRING[1];

{$V-,C-,R-} { Pascal Directives. See Compiler Manual }
  {$I TURPROCS.TUR  Include Procedures In This File. See Manual }

(*   This is a summary of the procedures in TURPROCS.TUR

PROCEDURE BEEP(BeepOn: BOOLEAN);              { Sound Beep if BeepOn=TRUE }
PROCEDURE CLEAR_KBD;                          { Clear Keyboard Buffer }
PROCEDURE COLOR(foregr,backgr:BYTE);          { Set Color }
PROCEDURE WRITEC(vtext: STR80);               { Write Chars Using Color }
FUNCTION  SET_MONITOR_TYPE: INTEGER;          { Determine Monitor Type }
{ Display A Screen Sculptor Screen }          { 2=Color, 3=Mono }
PROCEDURE DISPLAY_SCREEN(screen_name: STR80; VAR file_existSS: BOOLEAN);
{ Display And Get An Item From Screen. See Detailed Desription In Manual }
PROCEDURE GETITEM(
                COL,LIN,LEN :          BYTE;      { Column, Line, Length }
                ITYPE :                CHAR;      { Type= C, N, D, Y, M  }
            VAR WITEM :                STR80;     { Variable Name        }
                PICT :                 STR80;     { Picture X, U, L, 9, 8 # }
                ITEM_LOW,ITEM_HIGH :   STR80;     { Range - Numerics/Date Only}
            VAR RET :                  STR2;      { Returned Code        }
                RETRIEVE :             BOOLEAN;   { False=Disp Only, True=Get }
                FGR_COLOR,BGR_COLOR :  BYTE       { Colors Foregr, Backgr }
                ); EXTERN;
```

```
*)

PROCEDURE RET_STATUS;
{ Check Status Of Variable retSS and return a code in 'actionSS'   set 'varSS'
  This procedure is called immediately following GETITEM }

{ Input to this procedure:
  when retSS is length 1 the values are any of the ASCII chars
  when retSS is length 2 the values are uSS, lSS, puSS, pdSS, function keys
                                        dSS, rSS
                                        ( See CONST Section For Meanings ) }
{ Output:
  The following codes are returned in actionSS : nextSS, prevSS,
                                                  exitSS, staySS }
{ Based upon 'actionSS' this procedure will then set 'varSS' to an integer,
  which represents the next item (variable ) to get.  }
BEGIN
  last_field_actionSS:=staySS;
  actionSS:=nextSS; { Initialize Action Code }
  if retrieveSS then { Is retrieveSS TRUE? }
  begin
    if ord(retSS[0])=2 then { Is retSS length 2 ? }
    begin
      CASE retSS[2] of
      { Action to be taken depending on the last key pressed }
        uSS, lSS: actionSS:=prevSS;  { Up Key, Left Key }
        dSS, rSS: actionSS:=nextSS;  {Down Key, Right Key}
        puSS: actionSS:= { Page Up   }staySS;
        pdSS: actionSS:= { Page Down }staySS;
        f1SS: actionSS:= {execute} exitSS;          (**** exits screen ****)
        f2SS,f3SS,f4SS,f5SS,
        f6SS,f7SS,f8SS,f9SS,f10SS: actionSS:=staySS; { Function Key }
      END { Case ret };
    end else { retSS is length 1 }
    begin
      if retSS=escSS { Escape Key } then actionSS:=exitSS
    end;
  { Any other key not in the above list will keep actionSS=nextSS }
  end; {retrieveSS}
  CASE actionSS of
    staySS: ;
    nextSS: begin
              varSS:=varSS+1; if varSS>screen_fieldSS then varSS:=1;
              if last_fieldSS and retrieveSS then actionSS:=last_field_actionSS
            end;
    prevSS: begin varSS:=varSS-1; if varSS<1 then varSS:=screen_fieldSS end;
    exitSS: ;
  END; { CASE }
END; {PROCEDURE RET_STATUS}

PROCEDURE GETNUM(
                COL,LIN,LEN :          BYTE;     { Column, Line, Length }
                ITYPE :                CHAR;     { Type= C, N, D, Y, M  }
            VAR WITEM :                REAL;     { Numerci Variable Name }
```

```
                PICT :                    STR80;     { Picture X, U, L, 9, 8 # }
                ITEM_LOW,ITEM_HIGH :      REAL;      { Range - Numerics/Date Only}
            VAR RET :                     STR2;      { Returned Code          }
                RETRIEVE :                BOOLEAN;   { False=Disp Only, True=Get }
                FGR_COLOR,BGR_COLOR :     BYTE       { Colors Foregr, Backgr }
                );
{ This Procedure converts numeric to string before calling GETITEM }
{ It then converts the result back to numeric }
VAR numSS,numloSS,numhiSS: STR80; errorcodeSS,dec_posSS: INTEGER;
BEGIN
  { Get # of Decimal Positions }
  dec_posSS:=ord(pict[0])-pos('.',pict);
  { Convert item, low and high range to string }
  STR(witem:0:dec_posSS,numSS);
  STR(item_low:0:dec_posSS,numloSS);
  STR(item_high:0:dec_posSS,numhiSS);
  GETITEM(col,lin,len,itype,numSS,pict,numloSS,numhiSS,
          ret,retrieve,fgr_color,bgr_color);
  { Convert string to numeric item }
  VAL(numSS, witem, errorcodeSS);
END; { GETNUM }


(**************************** SET HEADER ****************************************)
(*                                                                              *)
(*      set global header parameters to the local input parameters values       *)
(*                                                                              *)
(********************************************************************************)

PROCEDURE SET_HEADER;    (* added *)
begin
     with info do                    {set header values}
          begin
                 nx      := trunc(points);
                 nz      := trunc(aslen);
                 w       := width;
                 dt      := tsamp*1.0e-9;
                 v       := vel*1.0e9;
                 sp      := calspace;
                { Xavg    := trunc(navg);}
                 stn     := trunc(station);
                { ln      := lenght;        }
                { scns    := trunc(scans); }
                 mrk     := trunc(mark);
                 zoffset := trunc(depthoff);
                 dcap    := datcap;
                 srcfile :=fin;
           END;
       char4 := printtrans;
       char5 := process;
       sparename1:=fname;                    {save fname for restoration}
       sparename2:=fin;                      {save fin for restoration}
       sparename3:=fout;                     {save fout for restoration}
end; {set header variables}
```

```
(****************************** SET VAR ****************************************)
(*                                                                              *)
(*    set program local variables to the global header parameter values        *)
(*                                                                              *)
(*******************************************************************************)
PROCEDURE SET_VAR;          (* added *)
    BEGIN
        with info do
          begin
                 points := nx;
                 aslen := nz;
                 width := w;
                 tsamp := dt*1.0e9;
                 vel := v*1.0e-9;
                 calspace := sp;
                 navg := Xavg;
                 station := stn;
                 lenght := ln;
                 scans := scns;
                 mark := mrk;
                 depthoff := zoffset;
                 datcap := dcap;
                 fin := srcfile;
      end;
      fname := sparename1;           {restore former file name}
      {fin := sparename2; }          {restore former file name}
      fout := sparename3;           {restore former file name}
      printtrans:= char4;
      process:=char5;
 END; {SET_VAR}


(*VAR { Variables Section For B:TRANSFER }  {move to declaration area}
   OPERATOR: STRING[20]; Fname: STRING[40]; FOUT: STRING[40];
   PRINTTRANS: STRING[1]; PROCESS: STRING[1];  *)

PROCEDURE INIT_VAR_1; { B:TRANSFER }
{ Initialize Variables To Default Values }
BEGIN
    OPERATOR:='MEL'; Fname:='c: data Demo'; FOUT:='c: data Demo';
    PRINTTRANS:='N'; PROCESS:='N';
END; { PROCEDURE INIT_VARS_1}

PROCEDURE SCREEN_1; { B:TRANSFER }
BEGIN

if screenSS<>1 then
begin
  screenSS:=1; screen_fieldSS:=5; varSS:=1;
  retrieveSS:=FALSE; last_fieldSS:=FALSE;
  DISPLAY_SCREEN('TRANSFER.SCR',file_existSS);  { Display Screen }
  if not file_existSS then
  begin gotoxy(1,1); write('Missing Screen TRANSFER') end;
end;
```

```
retSS:='';

{ Display Items. Change retrieveSS to TRUE and INPUT items}
REPEAT { until actionSS = exitSS }
  CASE varSS of
1:  GETITEM(58,12,20,'C',OPERATOR,
     'XXXXXXXXXXXXXXXXXXXX','','',retSS,retrieveSS,15,0);


(** modified to read data header and update display as filename is entered ***)
2:  BEGIN
    GETITEM(30,14,40,'C',Fname,
     'XXXXXXXXXXXXXXXXXXXXXXXXXXX','','',retSS,retrieveSS,15,0);
       if ord(retSS[1]) = 13
         then
           begin
                while pos(' ',fname)=1 do  {remove leading blanks}
                delete(fname,1,1);
                assign(datfile,fname+'.hdr');
                {$I-} reset(datfile) {$I+};
                OK := (IOresult = 0);
                if not OK
              then  begin             {cannot find header file}
                       gotoxy(2,25);
                       write('Cannot find header file ',fname,'             ');
                       delay(2000);
                       gotoxy(1,25);      {erase message}
                       write('                                       ');
                       write('                                       ');
                       varss:=2;
                    end
              else                     {header file is good}
                begin
                sparename1 := fname;
                sparename3 := fout;
                blockread(datfile,buff,1);    {read header information}
                move(buff,info,sizeof(info)); {copy header into info}
                close(datfile);       {the following update the display entries}
                set_var;                      {set variables with info}
                gotoxy(2,25);
                write('radar data file should be ',fin);
                delay(2000);
           repeat
                giveup := false;
                assign(datfile,fin);
                {$I-} reset(datfile) {$I+};
                OK := (IOresult = 0);
                close(datfile);
                if not OK
                   then  begin
                            gotoxy(1,25);
                            write('                                          ');
                            write('                                          ');
                            gotoxy(2,25);
```

```
                  write('Cannot find radar data file ',fin,' enter new name ');
                            read(fin); if fin = '' then giveup := true;
                            varss:=2;
                          end;
              until OK or giveup;
                  end;
                  gotoxy(1,25);
                  write('                                     ');
                  write('                                     ');
                  if ok then begin
                             gotoxy(1,25);
                             write(' ready to execute transfer ');
                            end;
                  retSS:='P';          {break recursion,change retss to dn arr}
                  retrieveSS:=false;           {set display mode in screen_1}
                  screen_1;                    {redisplay enteries}

             end;
      END;
  3:  GETITEM(30,16,40,'C',FOUT,
       'XXXXXXXXXXXXXXXXXXXXXXXXXXXX','','',retSS,retrieveSS,15,0);
  4:  begin
       GETITEM(40,18,1,'Y',PRINTTRANS,
       'U','','',retSS,retrieveSS,15,0);
       char3 := printtrans;                  {update char 3}
      end;
  5: begin
      GETITEM(40,20,1,'Y',PROCESS,
       'U','','',retSS,retrieveSS,15,0);
      char4 := process;                      {update char 4}
     end;
    END; { CASE }

(**********end of modification ************)

  if varSS=screen_fieldSS then last_fieldSS:=TRUE;
  RET_STATUS; { Check the code in  retSS . Set   varSS  and  actionSS  }

{ Check to see whether to switch retrieveSS to true }
if last_fieldSS and (not retrieveSS) then
begin
  retrieveSS:=TRUE; last_fieldSS:=FALSE; actionSS:=staySS; varSS:=1;
end else
  last_fieldSS:=FALSE;

UNTIL actionSS=exitSS
END; { PROCEDURE SCREEN_1 }


BEGIN { Main Test Program }
  vtypeSS:=SET_MONITOR_TYPE; { 2=Color, 3=Mono }
  screenSS:=0; use_screenSS:=1; BeepOnSS:=FALSE;  {no beep}
  INIT_VAR_1; { B:TRANSFER }
  if chain_return then set_var  {set menu entries to previous values}
```

```
                    else begin
                           char4 := printtrans;
                           char5 := process;
                         end;
REPEAT

  CASE use_screenSS of
 1:  begin
        SCREEN_1; { B:TRANSFER }
        if retSS = #27              { esc  pressed}
            then begin
                    Use_screenSS:=0; {if  esc  EXIT to select}
                    chain_return := false;{reset return from workhorse flag}
                    assign(workhorse,'select.chn');
                    chain(workhorse);        {transfer control to select}
               end
            else if (retSS[2]=f1SS)  and OK then{ F1  executes workhorse}
               begin
                while pos(' ',fout)=1 do  {remove leading blanks}
                delete(fout,1,1);
                set_header;
                      begin
                       { halt; }    {memory compiled exit}
                        assign(workhorse,'transfwk.chn');
                        chain(workhorse);      {execute transfwk.pas}
                      end;
               end
               else        {F1 but not OK}
                  begin
                    gotoxy(1,25);
                    write(' Data file ',fin,' not found                         ');
                    use_screenSS := 1;
                    varSS := 2;
                  end;
     end;
  ELSE
     use_screenSS:=0;
  END; { CASE }
UNTIL use_screenSS=0;

END. { PROGRAM TRANSFER }
```

```
(************************ T-SAFT MENU PROGRAM *********************************)
(*                                                                            *)
(*     This menu program has been generated by a  screen utility program      *)
(*  listed below. Modifications have been made of this program to add global *)
(*  variable definitions, variables value setting and resetting routines,     *)
(*  input checking codes and additional program flow software.  These         *)
(*  enhancements are made in all the menu programs to different degrees       *)
(*  depending on specific requirements.  This program has been well           *)
(*  documented here as well as in the screen sculptor manuals.                *)
(*                                                                            *)
(*     The function of this menu program is to provide a friendly user        *)
(*  interface for program control and parameter input. when this program is  *)
(*  executed predetermined values are given to these varisbles.  The          *)
(*  operator can change these values and is provided immediate feed back      *)
(*  on the screen.  In this program values presented on the screen takes      *)
(*  precedence over the program header values.  Displayed values can be       *)
(*  updated by simply depressing the carriage return when the cursor is at    *)
(*  the input file name.                                                      *)
(*                                                                            *)
(*                                                             7-1-87         *)
(******************************************************************************)

PROGRAM TSAFT(INPUT,OUTPUT);
{Procedures Generated By Screen Sculptor,Version 1.2 6/19/1987 - 21:3:3 }
CONST CopyrightSS='(C)Copyright 84,85 The Software Bottling Company Of New York';
{    DO NOT REMOVE The Above Copyright Notice
     This Program may not be used without the above Copyright Notice
}
{  SCREEN SCULPTOR(C) Version 1.2
   (C) COPYRIGHT, THE SOFTWARE BOTTLING COMPANY OF NEW YORK, 1984, 1985
   WARNING: Do not attempt to make any changes to this procedure unless you
   have made a backup. The Software Bottling Company Of New York can not
   and will not support such changes made to this program.
   Turbo(tm) Pascal Version, Trade Mark Of Borland International}

{ Global Required Definition of variables and constants }


    TYPE                              {writeheader variables}
          id = RECORD
                 nx,                  {nx= # of A-scans in one data set}
                 nz     :integer;     {nx = A-scan lenght, normally 400}
                 w,                   {w = survey width}
                 dt,                  {dt = sampling period}
                 v      :real;        {v = radar propagation velecity}
                 sp     :real;        {sp = averaged data scan-scan spacing}
                 Xavg   :integer;     {Xavg = # of A-scan averaged per data A-scan}
                 stn    :integer;     {station = data station}
                 ln     :real;        {ln = data lenght}
                 scns   :integer;     {scns =# of A-scansin the data}
                 mrk    :integer;     {delimiter between A-scans in original data}
                 zoffset:integer;     {template determined sample offset}
                 dcap   :real;        { data capacity of a disk }
                 srcfile:string[40];{scource data file}
                 spare1:real;       {template determined velocity}
                 spare2:integer;    {spare}
               END;
     VAR    { global variables }
            fname: string[40];          {name of I/O file}
            fin: string[40];            {name of input file}
            workhorse:file;             {chainfile assignee}
            fout:string[40];            {name of output file}
            which_screen:integer;       {which screen to use }
            chain_return:boolean;       {returning from chain program?}
            auto_processing:string[14];{automatic disk processing key}
            {the following characters are used for program flow control}
            char1,                      {plot intensity var in Preview}
            char2,                      {width calibtation var in Preview}
            char3,                      {print var in Preview}
            char4,                      {print var in Transfer}
            char5,                      {disk process var in Transfer}
            char6,                      {multiple processing var in T-saft}
            char7,                      {velocity stepping var in T-saft}
            char8,                      {print image var in T-saft}
            char9,                      {display format var in T-saft}
            char10,                  {spare character variables}
            char11,                  {spare character variables}
            char12,                  {spare character variables}
            char13,                  {spare character variables}
            char14,                  {spare character variables}
            char15:char;             {spare character variables}
            sparename1,
            sparename2,
            sparename3,
            sparename4:string[40];      {spare strings}
            spareI1,
            spareI2,
            spareI3,
            spare14:integer;            {spare integers}
            spareR1,
            spareR2,
            spareR3,
            spareR4:real;               {spare reals}
            info:id;                    {header information}

{the above are global variables to be passed from progran to program during
 the chain calls. These variables are to be declared exactly as above in all
 programs wether used or not.  Tailoring will be done with the spares.}


{what follows below are translation of the above header variables}

 VAR  points,aslen:real{integer};
      width,tsamp,vel,calspace:real;
      navg,station:real{integer};
      lenght:real;
      scans,mark:real{integer};
      spare:real;
```

```
        datcap:real;
        srcfile:string[14];
        detv:real;
        depthoff:real{integer};
        datafilenames1,datafilenames2:string[14];
        datfile:file;
        buff: ARRAY[0..127] OF byte;  {input buffer for blocking data and header}
        OK:boolean;                  {file existence flag}
        indexinc:integer;            {file index increment in multiproc}

TYPE    STR2 = STRING[2]; STR80 = STRING[80]; STR79 = STRING[79];
        resSS = (staySS, prevSS, exitSS, nextSS);
        string40 = string[40];           {added}

CONST { Esc, Up Arrow Key, Left Arrow Key , Page Up Key       }
        escSS=#27;   uSS='H'; lSS='K'; puSS='I';
        { Blank, Down Arrow Key, Right Arrow Key, Page Down Key }
        blankSS=' '; dSS='P'; rSS='M'; pdSS='Q';
        { Function keys  F1-F10 }
        f1SS=';'; f2SS='<'; f3SS='='; f4SS='>'; f5SS='?';
        f6SS=' '; f7SS='A'; f8SS='B'; f9SS='C'; f10SS='D';
        retSS : STR2='';

VAR     actionSS, last_field_actionSS : resSS;
        hiSS, loSS : REAL;
        vtypeSS, screenSS, use_screenSS, screen_fieldSS, varSS : INTEGER;
        file_existSS, last_fieldSS, retrieveSS : BOOLEAN;
        rangeSS : STR80;
        BeepOnSS: BOOLEAN;
        parsename: string[40];           {added}

VAR { Variables Section For B:T-SAFT }
      OPERATOR: STRING[20]; {FNAME: STRING[40]; VEL: REAL; DEPTHOFF: REAL;
      TSAMP: REAL; STATION: REAL;} SCALE: REAL; {CALSPACE: REAL; POINTS: REAL;
      WIDTH: REAL;} MULTIPROC: STRING[1]; STEP: STRING[1]; PRINT: STRING[1];
      DIS_FORM: STRING[1];

{$V-,C-,R-} { Pascal Directives. See Compiler Manual }
  {$I TURPROCS.TUR  Include Procedures In This File. See Manual }

(*   This is a summary of the procedures in TURPROCS.TUR

PROCEDURE BEEP(BeepOn: BOOLEAN);             { Sound Beep if BeepOn=TRUE }
PROCEDURE CLEAR_KBD;                         { Clear Keyboard Buffer }
PROCEDURE COLOR(foregr,backgr:BYTE);         { Set Color }
PROCEDURE WRITEC(vtext: STR80);              { Write Chars Using Color }
FUNCTION  SET_MONITOR_TYPE: INTEGER;         { Determine Monitor Type }
{ Display A Screen Sculptor Screen }         { 2=Color, 3=Mono }
PROCEDURE DISPLAY_SCREEN(screen_name: STR80; VAR file_existSS: BOOLEAN);
{ Display And Get An Item From Screen. See Detailed Desription In Manual }
PROCEDURE GETITEM(
                COL,LIN,LEN :          BYTE;     { Column, Line, Length }
                ITYPE :                CHAR;     { Type= C, N, D, Y, M  }
            VAR WITEM :                STR80;    { Variable Name        }
```

```
                PICT :                  STR80;     { Picture X, U, L, 9, 8 # }
                ITEM_LOW,ITEM_HIGH :    STR80;     { Range - Numerics/Date Only}
            VAR RET :                   STR2;      { Returned Code          }
                RETRIEVE :              BOOLEAN;   { False=Disp Only, True=Get }
                FGR_COLOR,BGR_COLOR :   BYTE       { Colors Foregr, Backgr }
                ); EXTERN;

*)

PROCEDURE RET_STATUS;
{ Check Status Of Variable retSS and return a code in 'actionSS'   set 'varSS'
  This procedure is called immediately following GETITEM }

{ Input to this procedure:
  when retSS is length 1 the values are any of the ASCII chars
  when retSS is length 2 the values are uSS, lSS, puSS, pdSS, function keys
                                         dSS, rSS
                                         ( See CONST Section For Meanings ) }
{ Output:
  The following codes are returned in actionSS : nextSS, prevSS,
                                                  exitSS, staySS }
{ Based upon 'actionSS' this procedure will then set 'varSS' to an integer,
  which represents the next item (variable ) to get.  }
BEGIN
  last_field_actionSS:=staySS;
  actionSS:=nextSS; { Initialize Action Code }
  if retrieveSS then { Is retrieveSS TRUE? }
  begin
    if ord(retSS[0])=2 then { Is retSS length 2 ? }
    begin
      CASE retSS[2] of
      { Action to be taken depending on the last key pressed }
        uSS, lSS: actionSS:=prevSS;  { Up Key, Left Key }
        dSS, rSS: actionSS:=nextSS;  {Down Key, Right Key}
        puSS: actionSS:= { Page Up   }staySS;
        pdSS: actionSS:= { Page Down }staySS;
        f1SS: actionSS:= {execute} exitSS;          (**** exits screen ****)
        f2SS,f3SS,f4SS,f5SS,
        f6SS,f7SS,f8SS,f9SS,f10SS: actionSS:=staySS; { Function Key }
      END { Case ret };
    end else { retSS is length 1 }
    begin
      if retSS=escSS { Escape Key } then actionSS:=exitSS
    end;
  { Any other key not in the above list will keep actionSS=nextSS }
  end; {retrieveSS}
  CASE actionSS of
    staySS: ;
    nextSS: begin
              varSS:=varSS+1; if varSS>screen_fieldSS then varSS:=1;
              if last_fieldSS and retrieveSS then actionSS:=last_field_actionSS
            end;
    prevSS: begin varSS:=varSS-1; if varSS<1 then varSS:=screen_fieldSS end;
    exitSS: ;
  END; { CASE }
END; {PROCEDURE RET_STATUS}

PROCEDURE GETNUM(
                COL,LIN,LEN :          BYTE;      { Column, Line, Length }
                ITYPE :                CHAR;      { Type= C, N, D, Y, M  }
            VAR WITEM :                REAL;      { Numerci Variable Name }
                PICT :                 STR80;     { Picture X, U, L, 9, 8 # }
                ITEM_LOW,ITEM_HIGH :   REAL;      { Range - Numerics/Date Only}
            VAR RET :                  STR2;      { Returned Code          }
                RETRIEVE :             BOOLEAN;   { False=Disp Only, True=Get }
                FGR_COLOR,BGR_COLOR :  BYTE       { Colors Foregr, Backgr }
                );
{ This Procedure converts numeric to string before calling GETITEM }
{ It then converts the result back to numeric }
VAR numSS,numloSS,numhiSS: STR80; errorcodeSS,dec_posSS: INTEGER;
BEGIN
  { Get # of Decimal Positions }
  dec_posSS:=ord(pict[0])-pos('.',pict);
  { Convert item, low and high range to string }
  STR(witem:0:dec_posSS,numSS);
  STR(item_low:0:dec_posSS,numloSS);
  STR(item_high:0:dec_posSS,numhiSS);
  GETITEM(col,lin,len,itype,numSS,pict,numloSS,numhiSS,
          ret,retrieve,fgr_color,bgr_color);
  { Convert string to numeric item }
  VAL(numSS, witem, errorcodeSS);
END; { GETNUM }


(**************************** SET HEADER *****************************************)
(*                                                                               *)
(*      set global header parameters to the local input parameters values        *)
(*                                                                               *)
(*********************************************************************************)

PROCEDURE SET_HEADER;   (* added *) {set header values to the input parameters}
begin
     with info do
          begin
                 nx      := trunc(points);
                 nz      := trunc(aslen);
                 w       := width;
                 dt      := tsamp*1.0e-9;    {note: vel and tsamp have their}
                 v       := vel*1.0e9;       {actual values in the header}
                 sp      := calspace;
               { Xavg    := trunc(navg);}
                 stn     := trunc(station);
               { ln      := lenght;        }
               { scns    := trunc(scans); }
                 mrk     := trunc(mark);
                 zoffset := trunc(depthoff);
                 dcap    := datcap;
                 srcfile :=srcfile;
            END;
        char6 := multiproc;
        char7 := step;
        char8 := print;
        char9 := dis_form;
        spareI1:=trunc(scale);
        sparename1:=fname;                {save fname for restoration}
end; {set header variables}


(***************************** SET VAR ****************************************)
(*                                                                            *)
(*  set program local variables to the global header parameter values         *)
(*                                                                            *)
(******************************************************************************)
PROCEDURE SET_VAR;       (* added *)  {set program values to the global}
    BEGIN                             {header values}
        with info do
          begin
                 points := nx;
                 aslen := nz;
                 width := w;
                 tsamp := dt*1.0e9;  {tsamp and vel units here are nS}
                 vel := v*1.0e-9;    {and ft/nS}
                 calspace := sp;
                 navg := Xavg;
                 station := stn;
                 lenght := ln;
                 scans := scns;
                 mark := mrk;
                 depthoff := zoffset;
                 datcap := dcap;
                 fin := srcfile;
      end;
      fname     := sparename1;            {restore former file name}
      multiproc :=char6;
      step      :=char7;
      print     :=char8;
      dis_form  :=char9;
      scale     :=spareI1;

 END; {SET_VAR}

(******************************** PARSE ***************************************)
(*                                                                            *)
(*     Procedure parses the name in fname into its path, rootname,index       *)
(*     and extension.  The index is incremented by indexinc value and         *)
(*     the fname is reassenbled with this new index.  The index will occupy  *)
(*     no more than the two last position of the eight character name.        *)
(*     This allow a miminum of six character for the name and a maximum       *)
(*     of 98 for the index value.                                             *)
(*                                                                            *)
(*     NOTE: File names are restricted to valid non numeric characters        *)
(*     only. Any number in the name will be interpreted as the index          *)
(*    of the name and be used to generate the resualtant name.           *)
(*                                                                        *)
(************************************************************************)

procedure parse(var fname:string40; var indexinc:integer);
var
  fpath,                    {path name}
  faux1,                    {root file name}
  faux2:string[40];         {intermediate path name segment}
  indst:string[2];          {string equvalent of index}
  chr1:char;                {temporary index character}
  chr2:char;                {                          }
  idx,                      {integer equ of index string}
  code:integer;             {integer conversion result}
begin
  fpath := '';
  faux1 := '';
  faux2 := '';
  indst := '';

  faux2 := copy(fname,1,pos(':',fname));
  fpath := concat(fpath,faux2);
writeln('drive = ',fpath);
  delete(fname,1,pos(':',fname));
repeat                                        {extract file path}
  faux2 := copy(fname,1,pos(' ',fname));  {identify directory levels}
  fpath := concat(fpath,faux2);           {add next path segment to path name}
  delete(fname,1,pos(' ',fname));         {remove path segment}
writeln(' path =',fpath);
until length(faux2) = 0;                      {no more path name segment}

  if pos('.',fname) <> 0
  then  faux1 := copy(fname,1,pos('.',fname)-1) {extract root file name}
  else  faux1 := fname;
  chr1 := copy(faux1,length(faux1),1);
  chr2 := copy(faux1,length(faux1)-1,1);
  if  (chr1 in ['0'..'9'])
    then if (chr2 in ['0'..'9'])
            then begin faux2 := copy(faux1,length(faux1)-1,2);
                       faux1 := copy(faux1,1,length(faux1)-2);
                 end
            else begin faux2 := copy(faux1,length(faux1),1);
                       faux1 := copy(faux1,1,length(faux1)-1);
                 end;
writeln(' root portion of fname ',faux1);
writeln(' index portion of fname ',faux2);
  val(faux2,idx,code);
  idx := idx + indexinc;
  str(idx,indst);                         {set indst to index value}
  fname := copy(faux1,1,8-length(indst));
  fname := concat(fpath,fname,indst,'.sft');
  indexinc := idx;
end;     {procedure parse}
```

```
(*VAR { Variables Section For B:T-SAFT } {moved up to declaration area}
   OPERATOR: STRING[20]; FNAME: STRING[40]; VEL: REAL; DEPTHOFF: REAL;
   TSAMP: REAL; STATION: REAL; SCALE: REAL; CALSPACE: REAL; POINTS: REAL;
   WIDTH: REAL; MULTIPROC: STRING[1]; STEP: STRING[1]; PRINT: STRING[1];
   DIS_FORM: STRING[1];
                                                                          *)
PROCEDURE INIT_VAR_1; { B:T-SAFT }
{ Initialize Variables To Default Values }
BEGIN
    OPERATOR:='MEL'; FNAME:='c: data Demo    .sft'; VEL:=0.4000;
    TSAMP:=0.1600; STATION:=0; SCALE:=20; CALSPACE:=0.3226; POINTS:=32;
    WIDTH:=30.000; MULTIPROC:='N'; STEP:='N'; PRINT:='N'; DIS_FORM:='W';
    DEPTHOFF:=0;
END; { PROCEDURE INIT_VARS_1}

PROCEDURE SCREEN_1; { B:T-SAFT }   {display the menu screen}
BEGIN

if screenSS<>1 then
begin
  screenSS:=1; screen_fieldSS:=14; varSS:=1;
  retrieveSS:=FALSE; last_fieldSS:=FALSE;
  DISPLAY_SCREEN('T-SAFT.SCR',file_existSS);  { Display Screen }
  if not file_existSS then
  begin gotoxy(1,1); write('Missing Screen T-SAFT') end;
end;
retSS:='';

{ Display Items. Change retrieveSS to TRUE and INPUT items}
REPEAT { until actionSS = exitSS }
  CASE varSS of
1:  GETITEM(55,12,20,'C',OPERATOR,
     'XXXXXXXXXXXXXXXXXXXX','','',retSS,retrieveSS,15,0);

(** modified to read data header and update display as filename is entered ***)
2:  BEGIN
       GETITEM(25,14,40,'C',FNAME,
       'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX','','',retSS,retrieveSS,15,0);
       gotoxy(1,25);
       write('  enter CR to update file parameters                           ');
       if ord(retSS[1]) = 13             {carriage return, seek file and}
         then                            {update parameters}
           begin
                while pos(' ',fname)=1 do  {remove leading blanks}
                delete(fname,1,1);
                assign(datfile,fname);
                {$I-} reset(datfile) {$I+};
                OK := (IOresult = 0);
                if not OK                    {unsuccessful open}
              then  begin
                       gotoxy(1,25);
                       write('                                        ');
                       write('                                        ');
                       gotoxy(1,25);
```

```
                    write(' Cannot find file ',fname);
                    delay(2000);          {clear error message}
                    gotoxy(1,25);
                    write('                                     ');
                    write('                                     ');
                    varss:=2;
                    screen_1;
                 end
            else                            {sucessful file opening}
             begin
              blockread(datfile,buff,1);    {read header information}
              move(buff,info,sizeof(info)); {copy header into info}
              close(datfile);     {the following update the display entries}
              char6 := multiproc;
              char7 := step;
              char8 := print;
              char9 := dis_form;
              spareI1:=trunc(scale);
              sparename1:=fname;
              set_var;                      {set variables with info}
              retSS:='P';           {break recursion,change retss to dn arr}
              retrieveSS:=false;            {set display mode in screen_1}
              gotoxy(1,25);
              write('  F1  to proceed, displayed parameter,',
              ' values overrides ');
              delay(3000);
              screen_1;                     {redisplay enteries}
             end;
          end;
    END;
          (** end of modification for case 2 **)

3:  GETNUM(25,16,7,'N',VEL,
     '##.####',0.0000,1.0000,retSS,retrieveSS,15,0);
4:  GETNUM(25,17,7,'N',DEPTHOFF,
     '#######',-400.0,400.0,retSS,retrieveSS,15,0);
5:  GETNUM(25,18,7,'N',TSAMP,
     '##.####',0.0000,2.0000,retSS,retrieveSS,15,0);
6:  GETNUM(25,19,7,'N',STATION,
     '#######',-999999.0,9999999.0,retSS,retrieveSS,15,0);
7:  GETNUM(28,20,4,'N',SCALE,
     '####',10.0,20.0,retSS,retrieveSS,15,0);
8:  GETNUM(25,21,7,'N',CALSPACE,
     '##.####',0.0000,2.0000,retSS,retrieveSS,15,0);
9:  GETNUM(59,16,7,'N',POINTS,
     '#######',2.0,32.0,retSS,retrieveSS,15,0);
10:  GETNUM(59,17,7,'N',WIDTH,
     '###.###',0.000,100.000,retSS,retrieveSS,15,0);
11:  GETITEM(74,18,1,'Y',MULTIPROC,
     'U','','',retSS,retrieveSS,15,0);
12:  GETITEM(74,19,1,'Y',STEP,
     'U','','',retSS,retrieveSS,15,0);
13:  GETITEM(74,20,1,'Y',PRINT,
     'U','','',retSS,retrieveSS,15,0);
```

```
14:  GETITEM(74,21,1,'C',DIS_FORM,
     'U','','',retSS,retrieveSS,15,0);
  END; { CASE }

  if varSS=screen_fieldSS then last_fieldSS:=TRUE;
  RET_STATUS; { Check the code in  retSS . Set   varSS  and  actionSS  }

{ Check to see whether to switch retrieveSS to true }
if last_fieldSS and (not retrieveSS) then
begin
  retrieveSS:=TRUE; last_fieldSS:=FALSE; actionSS:=staySS; varSS:=1;
end else
  last_fieldSS:=FALSE;

UNTIL actionSS=exitSS
END; { PROCEDURE SCREEN_1 }


BEGIN { Main Test Program }
  vtypeSS:=SET_MONITOR_TYPE; { 2=Color, 3=Mono }
  screenSS:=0; use_screenSS:=1; BeepOnSS:=false;   (* no beep *)
  INIT_VAR_1; { B:T-SAFT }
  if chain_return then set_var;     {set menu entries to previous values}
  indexinc := 1;
  if auto_processing = 'Autoprocess'
  then scale := 20;                 {set full vertical scale to 20 feet}

if (multiproc = 'Y') and chain_return then
            begin    {multiprocess}
               parse(fname,indexinc);
               if indexinc < 99 then      {check for filename index overflow}
               begin     {in range}
               assign(datfile,fname);
               {$I-} reset(datfile) {$I+};
               OK := (IOresult = 0);
               while Ok do
                 begin {sequential file processing}
                   blockread(datfile,buff,1);     {read header information}
                   move(buff,info,sizeof(info)); {copy header into info}
                   close(datfile);
                   station := info.stn;
                   set_header;
                      begin
                        (*gotoxy(1,25); write(' multiprocess halt ',fname,
                                        '                             ');
                        HALT;                   {test exit}  *)
                        assign(workhorse,'tsaftwk.chn');
                        chain(workhorse);
                      end;
                 end;       {sequential file processing}
                 end;     {in range}
                 multiproc := 'N';
                 gotoxy(1,25);
                 write('  next file ',fname,' not found, ');
```

```
                    if auto_processing = 'Autoprocess'
                        then write('** Auto Processing Completed ** ')
                        else write(' multi processing finished   ');
               end;       {multiprocess}
REPEAT
  CASE use_screenSS of
 1:  begin    {case1}
        SCREEN_1; { B:T-SAFT }
        if retSS = #27
            then begin  {esc}
                   assign(workhorse,'Select.chn');  {if  esc  EXIT to}
                   chain(workhorse);                {  Selection menu}
                   chain_return := false;
                 end    {esc}
            else if (retSS[2]=f1SS)  then   {if  F1  then execute workhorse}
               begin  {F1}
                while pos(' ',fname)=1 do  {remove leading blanks}
                delete(fname,1,1);
                while (pos(' ',fname)<>0) and (pos(' ',fname)<pos('.',fname))
                  do delete(fname,pos(' ',fname),1); {remove internal blanks}
                assign(datfile,fname);
                {$I-} reset(datfile) {$I+};
                OK := (IOresult = 0);
                if not OK                    {unsuccessful open}
              then  begin   {not ok}
                      gotoxy(2,25);
                      write('Cannot find file ',fname,'                    ');
                      delay(3000);           {clear error message}
                      gotoxy(1,25);
                      write('                                           ');
                   end     {not ok}
              else
                begin       {ok, single/first pass}
                  close(datfile);
                  set_header;
                      begin
                     (*   gotoxy(1,25); write('  single proc halt  ',fname,
                                        '                            ');
                        HALT;                  {test exit}*)
                        assign(workhorse,'tsaftwk.chn');
                        chain(workhorse);
                      end;
                 end;       {ok, single/first pass}
               end;   {F1}
     end;      {case1}
  ELSE
     use_screenSS:=0;
  END; { CASE }
UNTIL use_screenSS=0;

END. { PROGRAM B:TSAFT.PAS }
```

```
(************************* VELOCITY MENU PROGRAM ******************************)
(*                                                                            *)
(*     This menu program has been generated by a  screen utility program     *)
(*  listed below. Modifications have been made of this program to add global *)
(*  variable definitions, variables value setting and resetting routines,    *)
(*  input checking codes and additional program flow software.  These        *)
(*  enhancements are made in all the menu programs to different degrees      *)
(*  depending on specific requirements.  This program has been well          *)
(*  documented here as well as in the screen sculptor manuals.               *)
(*                                                                            *)
(*     The function of this menu program is to provide a friendly user       *)
(*  interface for program control and parameter input. when this program is  *)
(*  executed predetermined values are given to these varisbles.  The         *)
(*  operator can change these values and is provided immediate feed back     *)
(*  on the screen.  In this program values presented on the screen takes     *)
(*  precedence over the program header values.  Displayed values can be      *)
(*  updated by simply depressing the carriage return when the cursor is at   *)
(*  the input file name.                                                     *)
(*                                                                            *)
(*                                                    7-1-87                 *)
(******************************************************************************)

PROGRAM VELOCITY(INPUT,OUTPUT);
{Procedures Generated By Screen Sculptor,Version 1.2 6/19/1987 - 7:55:49 }
CONST CopyrightSS='(C)Copyright 84,85 The Software Bottling Company Of New York';
{    DO NOT REMOVE The Above Copyright Notice
     This Program may not be used without the above Copyright Notice
}
{  SCREEN SCULPTOR(C) Version 1.2
   (C) COPYRIGHT, THE SOFTWARE BOTTLING COMPANY OF NEW YORK, 1984, 1985
   WARNING: Do not attempt to make any changes to this procedure unless you
   have made a backup. The Software Bottling Company Of New York can not
   and will not support such changes made to this program.
   Turbo(tm) Pascal Version, Trade Mark Of Borland International}

{ Global Required Definition of variables and constants }

    TYPE                               {writeheader variables}
          id = RECORD
                 nx,                   {nx= # of A-scans in one data set}
                 nz     :integer;      {nx = A-scan lenght, normally 400}
                 w,                    {w = survey width}
                 dt,                   {dt = sampling period}
                 v      :real;         {v = radar propagation velecity}
                 sp     :real;         {sp = averaged data scan-scan spacing}
                 Xavg   :integer;      {Xavg = # of A-scan averaged per data A-scan}
                 stn    :integer;      {station = data station}
                 ln     :real;         {ln = data lenght}
                 scns   :integer;      {scns =# of A-scansin the data}
                 mrk    :integer;      {delimiter between A-scans in original data}
                 zoffset:integer;      {template determined sample offset}
                 dcap   :real;         { data capacity of a disk }
                 srcfile:string[40];{scource data file}
                 spare1:real;          {template determined velocity}
```

```
                spare2:integer;    {spare}
              END;
      VAR   { global variables }
            fname: string[40];       {name of I/O file}
            fin: string[40];         {name of input file}
            workhorse:file;          {chainfile assignee}
            fout:string[40];         {name of output file}
            which_screen:integer;    {which screen to use }
            chain_return:boolean;    {returning from chain program?}
            auto_processing:string[14];{automatic disk processing key}
            {the following characters are used for program flow control}
            char1,                   {plot intensity var in Preview}
            char2,                   {width calibtation var in Preview}
            char3,                   {print var in Preview}
            char4,                   {print var in Transfer}
            char5,                   {disk process var in Transfer}
            char6,                   {multiple processing var in T-saft}
            char7,                   {velocity stepping var in T-saft}
            char8,                   {print image var in T-saft}
            char9,                   {display format var in T-saft}
            char10,               {spare character variables}
            char11,               {spare character variables}
            char12,               {spare character variables}
            char13,               {spare character variables}
            char14,               {spare character variables}
            char15:char;          {spare character variables}
            sparename1,
            sparename2,
            sparename3,
            sparename4:string[40];   {spare strings}
            spareI1,
            spareI2,
            spareI3,
            spare14:integer;         {spare integers}
            spareR1,
            spareR2,
            spareR3,
            spareR4:real;            {spare reals}
            info:id;                 {header information}

   {the above are global variables to be passed from progran to program during
    the chain calls. These variables are to be declared exactly as above in all
    programs wether used or not.  Tailoring will be done with the spares.}

   {what follows below are translation of the above header variables}

    VAR  points,aslen:real{integer};
         width,tsamp,vel,calspace:real;
         navg,station:real{integer};
         lenght:real;
         scans,mark:real{integer};
         spare:real;
         datcap:real;
         srcfile:string[14];
```

```
        detv:real;
        depthoff:real{integer};
        datafilenames1,datafilenames2:string[14];
        datfile:file;
        buff: ARRAY[0..127] OF byte;  {input buffer for blocking data and header}
        OK:boolean;                  {file existence flag}
        giveup:boolean;              {flag for giving up looking for data file}


TYPE    STR2 = STRING[2]; STR80 = STRING[80]; STR79 = STRING[79];
        resSS = (staySS, prevSS, exitSS, nextSS);

CONST { Esc, Up Arrow Key, Left Arrow Key , Page Up Key        }
        escSS=#27;   uSS='H'; lSS='K'; puSS='I';
        { Blank, Down Arrow Key, Right Arrow Key, Page Down Key }
        blankSS=' '; dSS='P'; rSS='M'; pdSS='Q';
        { Function keys  F1-F10 }
        f1SS=';'; f2SS='<'; f3SS='='; f4SS='>'; f5SS='?';
        f6SS=' '; f7SS='A'; f8SS='B'; f9SS='C'; f10SS='D';
        retSS : STR2='';

VAR     actionSS, last_field_actionSS : resSS;
        hiSS, loSS : REAL;
        vtypeSS, screenSS, use_screenSS, screen_fieldSS, varSS : INTEGER;
        file_existSS, last_fieldSS, retrieveSS : BOOLEAN;
        rangeSS : STR80;
        BeepOnSS: BOOLEAN;

{$V-,C-,R-} { Pascal Directives. See Compiler Manual }
  {$I TURPROCS.TUR  Include Procedures In This File. See Manual }

(*   This is a summary of the procedures in TURPROCS.TUR

PROCEDURE BEEP(BeepOn: BOOLEAN);              { Sound Beep if BeepOn=TRUE }
PROCEDURE CLEAR_KBD;                          { Clear Keyboard Buffer }
PROCEDURE COLOR(foregr,backgr:BYTE);          { Set Color }
PROCEDURE WRITEC(vtext: STR80);               { Write Chars Using Color }
FUNCTION  SET_MONITOR_TYPE: INTEGER;          { Determine Monitor Type }
{ Display A Screen Sculptor Screen }          { 2=Color, 3=Mono }
PROCEDURE DISPLAY_SCREEN(screen_name: STR80; VAR file_existSS: BOOLEAN);
{ Display And Get An Item From Screen. See Detailed Desription In Manual }
PROCEDURE GETITEM(
                COL,LIN,LEN :          BYTE;      { Column, Line, Length }
                ITYPE :                CHAR;      { Type= C, N, D, Y, M  }
            VAR WITEM :                STR80;     { Variable Name        }
                PICT :                 STR80;     { Picture X, U, L, 9, 8 # }
                ITEM_LOW,ITEM_HIGH :   STR80;     { Range - Numerics/Date Only}
            VAR RET :                  STR2;      { Returned Code        }
                RETRIEVE :             BOOLEAN;   { False=Disp Only, True=Get }
                FGR_COLOR,BGR_COLOR :  BYTE       { Colors Foregr, Backgr }
                ); EXTERN;

*)
```

```
PROCEDURE RET_STATUS;
{ Check Status Of Variable retSS and return a code in 'actionSS'   set 'varSS'
  This procedure is called immediately following GETITEM }

{ Input to this procedure:
  when retSS is length 1 the values are any of the ASCII chars
  when retSS is length 2 the values are uSS, lSS, puSS, pdSS, function keys
                                        dSS, rSS
                                        ( See CONST Section For Meanings ) }
{ Output:
  The following codes are returned in actionSS : nextSS, prevSS,
                                                 exitSS, staySS }
{ Based upon 'actionSS' this procedure will then set 'varSS' to an integer,
  which represents the next item (variable ) to get.  }
BEGIN
  last_field_actionSS:=staySS;
  actionSS:=nextSS; { Initialize Action Code }
  if retrieveSS then { Is retrieveSS TRUE? }
  begin
    if ord(retSS[0])=2 then { Is retSS length 2 ? }
    begin
      CASE retSS[2] of
      { Action to be taken depending on the last key pressed }
        uSS, lSS: actionSS:=prevSS;  { Up Key, Left Key }
        dSS, rSS: actionSS:=nextSS;  {Down Key, Right Key}
        puSS: actionSS:= { Page Up   }staySS;
        pdSS: actionSS:= { Page Down }staySS;
        f1SS: actionSS:= {execute} exitSS;          (**** exits screen ****)
        f2SS,f3SS,f4SS,f5SS,
        f6SS,f7SS,f8SS,f9SS,f10SS: actionSS:=staySS; { Function Key }
      END { Case ret };
    end else { retSS is length 1 }
    begin
      if retSS=escSS { Escape Key } then actionSS:=exitSS
    end;
  { Any other key not in the above list will keep actionSS=nextSS }
  end; {retrieveSS}
  CASE actionSS of
    staySS: ;
    nextSS: begin
              varSS:=varSS+1; if varSS>screen_fieldSS then varSS:=1;
              if last_fieldSS and retrieveSS then actionSS:=last_field_actionSS
            end;
    prevSS: begin varSS:=varSS-1; if varSS<1 then varSS:=screen_fieldSS end;
    exitSS: ;
  END; { CASE }
END, {PROCEDURE RET_STATUS}

PROCEDURE GETNUM(
                COL,LIN,LEN :          BYTE;      { Column, Line, Length }
                ITYPE :                CHAR;      { Type= C, N, D, Y, M  }
            VAR WITEM :                REAL;      { Numerci Variable Name }
                PICT :                 STR80;     { Picture X, U, L, 9, 8 # }
                ITEM_LOW,ITEM_HIGH :   REAL;      { Range - Numerics/Date Only}
                VAR RET :                    STR2;      { Returned Code          }
                    RETRIEVE :               BOOLEAN;   { False=Disp Only, True=Get }
                    FGR_COLOR,BGR_COLOR : BYTE          { Colors Foregr, Backgr }
                    );
{ This Procedure converts numeric to string before calling GETITEM }
{ It then converts the result back to numeric }
VAR numSS,numloSS,numhiSS: STR80; errorcodeSS,dec_posSS: INTEGER;
BEGIN
  { Get # of Decimal Positions }
  dec_posSS:=ord(pict[0])-pos('.',pict);
  { Convert item, low and high range to string }
  STR(witem:0:dec_posSS,numSS);
  STR(item_low:0:dec_posSS,numloSS);
  STR(item_high:0:dec_posSS,numhiSS);
  GETITEM(col,lin,len,itype,numSS,pict,numloSS,numhiSS,
          ret,retrieve,fgr_color,bgr_color);
  { Convert string to numeric item }
  VAL(numSS, witem, errorcodeSS);
END; { GETNUM }


VAR { Variables Section For B:VEL }
   OPERATOR: STRING[20]; {FNAME: STRING[40];}

PROCEDURE INIT_VAR_1; { B:VEL }
{ Initialize Variables To Default Values }
BEGIN
    OPERATOR:='MEL'; FNAME:='c: data demo   .PRV';
END; { PROCEDURE INIT_VARS_1}

PROCEDURE SCREEN_1; { B:VEL }
BEGIN

if screenSS<>1 then
begin
  screenSS:=1; screen_fieldSS:=2; varSS:=1;
  retrieveSS:=FALSE; last_fieldSS:=FALSE;
  DISPLAY_SCREEN('VEL.SCR',file_existSS);  { Display Screen }
  if not file_existSS then
  begin gotoxy(1,1); write('Missing Screen VEL') end;
end;
retSS:='';

{ Display Items. Change retrieveSS to TRUE and INPUT items}
REPEAT { until actionSS = exitSS }
  CASE varSS of
1:  GETITEM(56,14,20,'C',OPERATOR,
      'XXXXXXXXXXXXXXXXXXXX','','',retSS,retrieveSS,15,0);
2:  GETITEM(31,16,40,'C',FNAME,
      'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX','','',retSS,retrieveSS,15,0);
  END; { CASE }

  if varSS=screen_fieldSS then last_fieldSS:=TRUE;
  RET_STATUS; { Check the code in  retSS . Set   varSS  and  actionSS  }
```

```
{ Check to see whether to switch retrieveSS to true }
if last_fieldSS and (not retrieveSS) then
begin
  retrieveSS:=TRUE; last_fieldSS:=FALSE; actionSS:=staySS; varSS:=1;
end else
  last_fieldSS:=FALSE;

UNTIL actionSS=exitSS
END; { PROCEDURE SCREEN_1 }


BEGIN { Main Test Program }
  vtypeSS:=SET_MONITOR_TYPE; { 2=Color, 3=Mono }
  screenSS:=0; use_screenSS:=1; BeepOnSS:=false; {no beep}
  INIT_VAR_1; { B:VEL }
  if chain_return then     {set menu entries to previous values}
                    begin
                      fname := sparename1;
                    end;
REPEAT

  CASE use_screenSS of
 1:  begin
        SCREEN_1; { B:VEL }

        if retSS = #27
            then begin
                    Use_screenSS:=0; {if  esc  EXIT to select}
                    chain_return := false;
                    assign(workhorse,'select.chn');
                    chain(workhorse);     {return control to select}
                 end
            else if (retSS[2]=f1SS)  then   {if  F1  then execute workhorse}
               begin
                while pos(' ',fname)=1 do  {remove leading blanks}
                delete(fname,1,1);
                assign(datfile,fname);
                {$I-} reset(datfile) {$I+}; {open data file}
                OK := (IOresult = 0);
             if not OK
              then  begin            {cannot find header file}
                       gotoxy(2,25);
                       write('Cannot find file ',fname,'            ');
                       delay(3000);
                       gotoxy(1,25);
                       write('                                                ');
                       varss:=2;
                       use_screenSS := 1;
                    end
                else                      {header file is good}
                  begin
                     close(datfile);
                     sparename1 := fname;  {save fname}
                     assign(workhorse,'velociwk.chn');
                    chain(workhorse);     {transfer control to workhorse}
                 end;
              end;
    end
 ELSE
    use_screenSS:=0;
 END; { CASE }
UNTIL use_screenSS=0;
END. { PROGRAM VELOCITY.PAS }
```

```
(************************ REVIEW MENU PROGRAM ********************************)
(*                                                                           *)
(*     This menu program has been generated by a  screen utility program     *)
(*  listed below. Modifications have been made of this program to add global *)
(*  variable definitions, variables value setting and resetting routines,    *)
(*  input checking codes and additional program flow software.  These        *)
(*  enhancements are made in all the menu programs to different degrees      *)
(*  depending on specific requirements.  This program has been well          *)
(*  documented here as well as in the screen sculptor manuals.               *)
(*                                                                           *)
(*     The function of this menu program is to provide a friendly user       *)
(*  interface for program control and parameter input. when this program is  *)
(*  executed predetermined values are given to these varisbles.  The         *)
(*  operator can change these values and is provided immediate feed back     *)
(*  on the screen.  In this program values presented on the screen takes     *)
(*  precedence over the program header values.  Displayed values can be      *)
(*  updated by simply depressing the carriage return when the cursor is at   *)
(*  the input file name.                                                     *)
(*                                                                           *)
(*                                                         7-1-87            *)
(*****************************************************************************)

PROGRAM REVIEW(INPUT,OUTPUT);
{Procedures Generated By Screen Sculptor,Version 1.2 6/30/1987 - 11:41:7 }
CONST CopyrightSS='(C)Copyright 84,85 The Software Bottling Company Of New York';
{    DO NOT REMOVE The Above Copyright Notice
     This Program may not be used without the above Copyright Notice
}
{  SCREEN SCULPTOR(C) Version 1.2
   (C) COPYRIGHT, THE SOFTWARE BOTTLING COMPANY OF NEW YORK, 1984, 1985
   WARNING: Do not attempt to make any changes to this procedure unless you
   have made a backup. The Software Bottling Company Of New York can not
   and will not support such changes made to this program.
   Turbo(tm) Pascal Version, Trade Mark Of Borland International}

{ Global Required Definition of variables and constants }

    TYPE                             {writeheader variables}
          id = RECORD
                 nx,                 {nx= # of A-scans in one data set}
                 nz     :integer;    {nx = A-scan lenght, normally 400}
                 w,                  {w = survey width}
                 dt,                 {dt = sampling period}
                 v      :real;       {v = radar propagation velecity}
                 sp     :real;       {sp = averaged data scan-scan spacing}
                 Xavg   :integer;    {Xavg = # of A-scan averaged per data A-scan}
                 stn    :integer;    {station = data station}
                 ln     :real;       {ln = data lenght}
                 scns   :integer;    {scns =# of A-scansin the data}
                 mrk    :integer;    {delimiter between A-scans in original data}
                 zoffset:integer;    {template determined sample offset}
                 dcap   :real;       { data capacity of a disk }
                 srcfile:string[40];{scource data file}
                 spare1:real;        {template determined velocity}
```

```
                spare2:integer;   {spare}
              END;
     VAR    { global variables }
          fname: string[40];        {name of I/O file}
          fin: string[40];          {name of input file}
          workhorse:file;           {chainfile assignee}
          fout:string[40];          {name of output file}
          which_screen:integer;     {which screen to use }
          chain_return:boolean;     {returning from chain program?}
          auto_processing:string[14];{automatic disk processing key}
          {the following characters are used for program flow control}
          char1,                    {plot intensity var in Preview}
          char2,                    {width calibtation var in Preview}
          char3,                    {print var in Preview}
          char4,                    {print var in Transfer}
          char5,                    {disk process var in Transfer}
          char6,                    {multiple processing var in T-saft}
          char7,                    {velocity stepping var in T-saft}
          char8,                    {print image var in T-saft}
          char9,                    {display format var in T-saft}
          char10,               {spare character variables}
          char11,               {spare character variables}
          char12,               {spare character variables}
          char13,               {spare character variables}
          char14,               {spare character variables}
          char15:char;          {spare character variables}
          sparename1,
          sparename2,
          sparename3,
          sparename4:string[40];    {spare strings}
          spareI1,
          spareI2,
          spareI3,
          spare14:integer;          {spare integers}
          spareR1,
          spareR2,
          spareR3,
          spareR4:real;             {spare reals}
          info:id;                  {header information}

 {the above are global variables to be passed from progran to program during
  the chain calls. These variables are to be declared exactly as above in all
  programs wether used or not.  Tailoring will be done with the spares.}
     var
          datfile:file;             {file variable}
          OK:boolean;               {file existence flag}
          extname:string[4];        {file name extension}
 TYPE  STR2 = STRING[2]; STR80 = STRING[80]; STR79 = STRING[79];
       resSS = (staySS, prevSS, exitSS, nextSS);

 CONST { Esc, Up Arrow Key, Left Arrow Key , Page Up Key       }
       escSS=#27;   uSS='H'; lSS='K'; puSS='I';
       { Blank, Down Arrow Key, Right Arrow Key, Page Down Key }
       blankSS=' '; dSS='P'; rSS='M'; pdSS='Q';
```

```
        { Function keys  F1-F10 }
        f1SS=';'; f2SS='<'; f3SS='='; f4SS='>'; f5SS='?';
        f6SS=' '; f7SS='A'; f8SS='B'; f9SS='C'; f10SS='D';
        retSS : STR2='';

VAR     actionSS, last_field_actionSS : resSS;
        hiSS, loSS : REAL;
        vtypeSS, screenSS, use_screenSS, screen_fieldSS, varSS : INTEGER;
        file_existSS, last_fieldSS, retrieveSS : BOOLEAN;
        rangeSS : STR80;
        BeepOnSS: BOOLEAN;

{$V-,C-,R-} { Pascal Directives. See Compiler Manual }
  {$I TURPROCS.TUR  Include Procedures In This File. See Manual }

(*  This is a summary of the procedures in TURPROCS.TUR

PROCEDURE BEEP(BeepOn: BOOLEAN);              { Sound Beep if BeepOn=TRUE }
PROCEDURE CLEAR_KBD;                          { Clear Keyboard Buffer }
PROCEDURE COLOR(foregr,backgr:BYTE);          { Set Color }
PROCEDURE WRITEC(vtext: STR80);               { Write Chars Using Color }
FUNCTION  SET_MONITOR_TYPE: INTEGER;          { Determine Monitor Type }
{ Display A Screen Sculptor Screen }          { 2=Color, 3=Mono }
PROCEDURE DISPLAY_SCREEN(screen_name: STR80; VAR file_existSS: BOOLEAN);
{ Display And Get An Item From Screen. See Detailed Desription In Manual }
PROCEDURE GETITEM(
                COL,LIN,LEN :          BYTE;      { Column, Line, Length }
                ITYPE :                CHAR;      { Type= C, N, D, Y, M  }
            VAR WITEM :                STR80;     { Variable Name        }
                PICT :                 STR80;     { Picture X, U, L, 9, 8 # }
                ITEM_LOW,ITEM_HIGH :   STR80;     { Range - Numerics/Date Only}
            VAR RET :                  STR2;      { Returned Code        }
                RETRIEVE :             BOOLEAN;   { False=Disp Only, True=Get }
                FGR_COLOR,BGR_COLOR :  BYTE       { Colors Foregr, Backgr }
                ); EXTERN;

*)

PROCEDURE RET_STATUS;
{ Check Status Of Variable retSS and return a code in 'actionSS'   set 'varSS'
  This procedure is called immediately following GETITEM }

{ Input to this procedure:
  when retSS is length 1 the values are any of the ASCII chars
  when retSS is length 2 the values are uSS, lSS, puSS, pdSS, function keys
                                         dSS, rSS
                                         ( See CONST Section For Meanings ) }
{ Output:
  The following codes are returned in actionSS : nextSS, prevSS,
                                                  exitSS, staySS }
{ Based upon 'actionSS' this procedure will then set 'varSS' to an integer,
  which represents the next item (variable ) to get.  }
BEGIN
  last_field_actionSS:=staySS;
```

```
    actionSS:=nextSS; { Initialize Action Code }
    if retrieveSS then { Is retrieveSS TRUE? }
    begin
      if ord(retSS[0])=2 then { Is retSS length 2 ? }
      begin
        CASE retSS[2] of
        { Action to be taken depending on the last key pressed }
          uSS, lSS: actionSS:=prevSS;  { Up Key, Left Key }
          dSS, rSS: actionSS:=nextSS;  {Down Key, Right Key}
          puSS: actionSS:= { Page Up   }staySS;
          pdSS: actionSS:= { Page Down }staySS;
          f1SS: actionSS:= {execute} exitSS;          (**** exits screen ****)
          f2SS,f3SS,f4SS,f5SS,
          f6SS,f7SS,f8SS,f9SS,f10SS: actionSS:=staySS; { Function Key }
        END { Case ret };
      end else { retSS is length 1 }
      begin
        if retSS=escSS { Escape Key } then actionSS:=exitSS
      end;
    { Any other key not in the above list will keep actionSS=nextSS }
    end; {retrieveSS}
    CASE actionSS of
      staySS: ;
      nextSS: begin
                varSS:=varSS+1; if varSS>screen_fieldSS then varSS:=1;
                if last_fieldSS and retrieveSS then actionSS:=last_field_actionSS
              end;
      prevSS: begin varSS:=varSS-1; if varSS<1 then varSS:=screen_fieldSS end;
      exitSS: ;
    END; { CASE }
  END; {PROCEDURE RET_STATUS}

  PROCEDURE GETNUM(
                  COL,LIN,LEN :         BYTE;     { Column, Line, Length }
                  ITYPE :               CHAR;     { Type= C, N, D, Y, M  }
              VAR WITEM :               REAL;     { Numerci Variable Name }
                  PICT :                STR80;    { Picture X, U, L, 9, 8 # }
                  ITEM_LOW,ITEM_HIGH :  REAL;     { Range - Numerics/Date Only}
              VAR RET :                 STR2;     { Returned Code          }
                  RETRIEVE :            BOOLEAN;  { False=Disp Only, True=Get }
                  FGR_COLOR,BGR_COLOR : BYTE      { Colors Foregr, Backgr }
                  );
  { This Procedure converts numeric to string before calling GETITEM }
  { It then converts the result back to numeric }
  VAR numSS,numloSS,numhiSS: STR80; errorcodeSS,dec_posSS: INTEGER;
  BEGIN
    { Get # of Decimal Positions }
    dec_posSS:=ord(pict[0])-pos('.',pict);
    { Convert item, low and high range to string }
    STR(witem:0:dec_posSS,numSS);
    STR(item_low:0:dec_posSS,numloSS);
    STR(item_high:0:dec_posSS,numhiSS);
    GETITEM(col,lin,len,itype,numSS,pict,numloSS,numhiSS,
            ret,retrieve,fgr_color,bgr_color);
  { Convert string to numeric item }
  VAL(numSS, witem, errorcodeSS);
END; { GETNUM }


VAR { Variables Section For A:REVIEW }
   {FNAME: STRING[40];} PRV: STRING[1]; SCR: STRING[1]; IMG: STRING[1];
   HDR: STRING[1]; SFT: STRING[1];

PROCEDURE INIT_VAR_1; { A:REVIEW }
{ Initialize Variables To Default Values }
BEGIN
    FNAME:='c: data demo'; PRV:=''; SCR:=''; IMG:=''; HDR:=''; SFT:='';
END; { PROCEDURE INIT_VARS_1}

PROCEDURE SCREEN_1; { A:REVIEW }
BEGIN

if screenSS<>1 then
begin
  screenSS:=1; screen_fieldSS:=6; varSS:=1;
  retrieveSS:=FALSE; last_fieldSS:=FALSE;
  DISPLAY_SCREEN('REVIEW.SCR',file_existSS);  { Display Screen }
  if not file_existSS then
  begin gotoxy(1,1); write('Missing Screen REVIEW') end;
end;
retSS:='';

{ Display Items. Change retrieveSS to TRUE and INPUT items}
REPEAT { until actionSS = exitSS }
  CASE varSS of          {display menu variables, and select appropriate}
1:  GETITEM(25,14,40,'C',FNAME, {control and file name extentions}
     'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX','','',retSS,retrieveSS,15,0);
2:  begin  GETITEM(8,18,1,'C',PRV,
            'X','','',retSS,retrieveSS,15,0);
           extname := '.prv';
           char10 := 'S';                { S  for displaying screen}
    end;
3:  begin  GETITEM(8,19,1,'C',SCR,
            'X','','',retSS,retrieveSS,15,0);
           extname := '.scr';
           char10 := 'S';                { S  for displaying screen}
    end;
4:  begin  GETITEM(8,20,1,'C',IMG,
            'X','','',retSS,retrieveSS,15,0);
           extname := '.img';
           char10 := 'S';               { S  for displaying screen}
    end;
5:  begin  GETITEM(8,21,1,'C',HDR,
            'X','','',retSS,retrieveSS,15,0);
           extname := '.hdr';
           char10 := 'H';                { H  for displaying header parameters}
    end;
6:  begin  GETITEM(8,22,1,'C',SFT,
             'X','','',retSS,retrieveSS,15,0);
            extname := '.sft';
            char10 := 'H';               { H  for displaying header parameters}
       end;
     END; { CASE }

     if varSS=screen_fieldSS then last_fieldSS:=TRUE;
     RET_STATUS; { Check the code in  retSS . Set   varSS  and  actionSS  }

   { Check to see whether to switch retrieveSS to true }
   if last_fieldSS and (not retrieveSS) then
   begin
     retrieveSS:=TRUE; last_fieldSS:=FALSE; actionSS:=staySS; varSS:=1;
   end else
     last_fieldSS:=FALSE;

   UNTIL actionSS=exitSS
   END; { PROCEDURE SCREEN_1 }

   BEGIN { Main Test Program }
     vtypeSS:=SET_MONITOR_TYPE; { 2=Color, 3=Mono }
     screenSS:=0; use_screenSS:=1; BeepOnSS:=false; {no beep}
     INIT_VAR_1; { A:REVIEW }
     if chain_return then      {set fname to previous values}
                       begin
                         fname := sparename1;
                       end;

   REPEAT

     CASE use_screenSS of
    1:  begin
           SCREEN_1; { B:VEL }

           if retSS = #27
               then begin
                       Use_screenSS:=0; {if  esc  EXIT to select}
                       chain_return := false;
                       assign(workhorse,'select.chn');
                       chain(workhorse);     {return control to select}
                    end
               else if (retSS[2]=f1SS)  then   {if  F1  then execute workhorse}
                  begin
                   while pos(' ',fname)=1 do  {remove leading blanks}
                   delete(fname,1,1);
                   sparename1 := fname;
                   if pos('.',fname) <> 0     {use extension if supplied}
                   then fname := fname
                   else fname := fname+extname; {otherwise use appropriate ext}
                   assign(datfile,fname);
                   {$I-} reset(datfile) {$I+};
                   OK := (IOresult = 0);
                 if not OK
                  then  begin            {cannot find file}
```

```
                         gotoxy(2,25);
                         write('Cannot find file ',fname,'           ');
                         fname := copy(fname,1,pos('.',fname)-1);
                         delay(3000);          {display and erase error message}
                         gotoxy(1,25);
                         write('                                            ');
                         varss:=2;
                         use_screenSS := 1;
                      end
                   else                        {file is good}
                     begin
                        close(datfile);
                        assign(workhorse,'reviewk.chn');
                        chain(workhorse);       {go to working program reviewk}
                     end;
                  end;
        end
     ELSE
        use_screenSS:=0;
     END; { CASE }
   UNTIL use_screenSS=0;            {redisplay screen until valid input or  esc }

   END. { PROGRAM A:REVIEW.PAS }
```

```
(********************** RADAR DATA PREVIEW PROGRAM ***************************)
(*                                                                            *)
(*    This program will read byte typed serial data of aslen per trace and    *)
(* plot data in 4 grey levels on the screen.  The plot width can be a         *)
(* percentage of disk capacity or scaled to full screen.  Data parameters     *)
(* can be input using the menu if the width and station calibration routine   *)
(* is not invoked, or additional trimming can be done using the calibration   *)
(* routine.  A negative station entry means that survey is done towards       *)
(* smaller station marking.  Station entries or calibration can be repeated,  *)
(* however only the latest parameters are retained.  The program works        *)
(* with a subsequent data selection program to extract 10 ft worth (or less) *)
(* of data for imaging.                                                       *)
(*    Outputs are a screen file with '.prv' extension and a header file       *)
(* with a '.hdr' extension.  Output files will only be generated if an        *)
(* output file name has been entered otherwise only the data plot is shown.   *)
(*    This program has been extended to work in the automatic whole disk      *)
(* processing mode.  In this mode control is passed to Transfer when          *)
(* preview is done.                                                           *)
(*                                                             7-1-87         *)
(****************************************************************************)

Program Preview;

{ Global Required Definiticn of variables and constants }


    TYPE                                {writeheader variables}
          id = RECORD
                 nx,                    {nx= # of A-scans in one data set}
                 nz     :integer;       {nx = A-scan lenght, normally 400}
                 w,                     {w = survey width}
                 dt,                    {dt = sampling period}
                 v      :real;          {v = radar propagation velecity}
                 sp     :real;          {sp = averaged data scan-scan spacing}
                 Xavg   :integer;       {Xavg = # of A-scan averaged per data A-scan}
                 stn    :integer;       {station = data station}
                 ln     :real;          {ln = data lenght}
                 scns   :integer;       {scns =# of A-scansin the data}
                 mrk    :integer;       {delimiter between A-scans in original data}
                 zoffset:integer;       {template determined sample offset}
                 dcap   :real;          { data capacity of a disk }
                 srcfile:string[40];{scource data file}
                 spare1:real;           {template determined velocity}
                 spare2:integer;        {spare}
               END;
    VAR   { global variables }
          fname: string[40];            {name of I/O file}
          fin: string[40];              {name of input file}
          workhorse:file;               {chainfile assignee}
          fout:string[40];              {name of output file}
          which_screen:integer;         {which screen to use }
          chain_return:boolean;         {returning from chain program?}
          auto_processing:string[14];{automatic disk processing key}
          {the following characters are used for program flow control}
```

```
            char1,                  {plot intensity var in Preview}
            char2,                  {width calibtation var in Preview}
            char3,                  {print var in Preview}
            char4,                  {print var in Transfer}
            char5,                  {disk process var in Transfer}
            char6,                  {multiple processing var in T-saft}
            char7,                  {velocity stepping var in T-saft}
            char8,                  {print image var in T-saft}
            char9,                  {display format var in T-saft}
            char10,               {spare character variables}
            char11,               {spare character variables}
            char12,               {spare character variables}
            char13,               {spare character variables}
            char14,               {spare character variables}
            char15:char;          {spare character variables}
            sparename1,
            sparename2,
            sparename3,
            sparename4:string[40];  {spare strings}
            spareI1,
            spareI2,
            spareI3,
            spare14:integer;        {spare integers}
            spareR1,
            spareR2,
            spareR3,
            spareR4:real;           {spare reals}
            info:id;                {header information}

{the above are global variables to be passed from progran to program during
 the chain calls. These variables are to be declared exactly as above in all
 programs wether used or not.  Tailoring will be done with the spares.}

type num = byte;
    heappoint= integer;             {heap pointer}

CONST
      nxmax = 64;                   {max no of ascans per dataset}
      {aslen = 400; }               {no of data points per ascan}
VAR
    {  fname: string[40]; }           {name of output file}
    {  fin: string[40]:   }           {name of input file}
    {  fout: string[40];  }           {output header file name}
      scrname : string[40];         {name of output preview screen file}
      A: array[1..400] of Num;      {byte input data buffer}
      b: array[1..400] of integer;  {integer plot data buffer}
      fdata:FILE;                   {output disk file}
      datfile: file of byte;        {raw data disk file}
      calibrate,                    {control for data width calibration}
      printprv,                     {control for data plot printing}
      intensity: char;              {selector for hi or lo intisity plot}
      printscr:boolean;             {control variable for plot printing}
      aslen,                        {length of radar traces, normally 400}
      n,k,i,j,l,m,                  {general indices}
      traces,                        { # lines in data plot}
      marker,                        {delimiter between A-sacn traces}
      navg,                          {# radar scans to average for data set}
      xpos: integer;                 {screen x coordinate }
      points,                        {number of A-scans per data set}
      grlevels,                      {# of grey levels in plot}
      markpos,                       {position of first data point}
      station:integer;               {ground station reference}
      benchmark:real;                {real equivalent of station}
      scans,                         {number of scanes}
      int_scale:integer;             {intensity of plot (hi or lo)}
      avgplot : real;                {#data scans to average for one plot line}
      datcap,                        {data disk capacity}
      avgr,                          {#data scan to average for data set}
      space,                         {A-scan spacing}
      width,                         {approx. survey width}
      lenght,                        {length in bytes of radar data file}
      readpos,                       {where to read data for plotting}
      calspace:real;                 {calculated scan spacing for saft data file}
      vel,                           {velocity}
      detv,                          {Velocity pgm determined velocity}
      tsamp: real;                   {data sampling period}
      depthoff:integer;              {depth offset in samples}
      heaptop:heappoint;             {pointer to the heap}


{$R+}
{$V-}

{These include files are defined in the Borland Turbo Graphix Toolbox
 manual and must be included in this order.  They are used in this program
 by procedure calls coded in the program.  For further information consult
 the Borland Turbo Graphix Toolbox manual.}
  {$I c: graphix typedef.sys}                      {variable declaration}
  {$I c: graphix graphix.sys}                      {definitions and routines}
  {$I c: graphix kernel.sys}                       {support routine}
  {$I c: graphix windows.sys}                      {graphics window manipulation
                                                           routine}
(************************** TIME and DATE *********************************)
(*                                                                        *)
(*                              time and date                             *)
(*                                                                        *)
(**************************************************************************)
procedure timedate(var hour,min,month,day,year:integer);
type
  registers = record
               ax,bx,cx,dx,bp,si,di,ds,es,flags:integer;
             end;
var
  regs: registers;

begin
  with regs do            {time of day}
```

```
    begin
      ax := $2C00;
      msdos(regs);          {MS DOS function call to DOS time function}
      hour := hi(cx);
      min := lo(cx);
    end;
  with regs do              {date}
    begin
      ax := $2A00;
      msdos(regs);          {MS DOS function call to DOS date function}
      year := cx;
      month := hi(dx);
      day := lo(dx);
    end;
end;    {timedate}

(******************************** BEEP *************************************)
(*                                                                         *)
(*                            makes a sound                                *)
(*                                                                         *)
(***************************************************************************)
procedure beep(freq:integer);
begin
  sound(freq);
  for i := 1 to 10000 do i:=i;
  nosound;
end;    {beep}


(********************************** ANNOTATE *********************************)
(*                                                                           *)
(*                 write pertinent parameters to the screen                  *)
(*                                                                           *)
(*****************************************************************************)
procedure annotate;
var
  hour,min,month,day,year:integer;

begin
  timedate(hour,min,month,day,year);
  gotoxy(1,25);
  write('                                                       '+
        '                           ');
  gotoxy(2,25);
  write(fin);
  gotoxy(15,25);
  write('X=',navg);
  gotoxy(20,25);
  write('W =',width:4:1);
  gotoxy(30,25);
  write('Sp=',Calspace:5:3);
  gotoxy(40,25);
  write('T=',tsamp*1E09:6:4);
  gotoxy(50,25);
  write(hour:2,':',min:2,' ',month:2,'-',day:2,'-',year:2);
  gotoxy(68,25);
  write('DL=',lenght:6:0);
  copyscreen;
end;    {annotate}



(********************************** SETUP **************************************)
(*                                                                             *)
(*       defines the display areas and the plotting scales [world]             *)
(*                                                                             *)
(*******************************************************************************)
Procedure setup;
var
  st:string[6];

begin
  clrscr;
  InitGraphic;
  DefineWindow(1,0,0,79,180);              {define data window size}
  Defineworld(1,0,0,639,aslen);            {coordinates limits}
  DefineWindow(2,0,180,79,199);            {degine axis window size}
  Defineworld(2,0,0,639,20);
  SelectWorld(2);                          {chose axis drawing window}
  SelectWindow(2);
  drawtextW(0,13,1,'0');
  drawtextW(610,13,1,'scans  ');           {annotate axis}
  for i := 0 to 9 do
    drawline((i)*64,17,(i)*64,20);         {draw 10 tick marks along bottom}
  for i := 1 to 9 do
    begin
      str(i*datcap*0.1/aslen:4:0,st);
      drawtextW(i*64-15,13,1,st);          {write scan scale}
    end;
  SelectWorld(1);                          {select data drawing area}
  SelectWindow(1);
  SetBackground(0);
  Drawborder;
  setcolorwhite;
end;     {setup}


(*********************************** PLOT **************************************)
(*                                                                             *)
(*    Makes line by line plot of raw data, a total of 160 lines                *)
(*    ( = 640/ 4 greylevels ) are plotted.                                     *)
(*    In this graphics version y=0 is at the bottom                            *)
(*                                                                             *)
(*******************************************************************************)
Procedure plot;
begin
  xpos :=(k)*grlevels;                                {center next trace}
  FOR i:=1 TO aslen DO
    begin
      n:= round((b[i]-127)*int_scale*grlevels/128);   {scales magnitude}
      if n > grlevels then n:=grlevels;               {max limiting}
      if n < -grlevels then n:= -grlevels;            {min limiting}
      drawline(xpos,aslen-i,xpos+n,aslen-i);          {draw a point}
    end;    {FOR}
end;    {PLOT}


(****************************** WRITEHEADER *********************************)
(*                                                                          *)
(*      write pertinent data parameters in data file header                 *)
(*                                                                          *)
(****************************************************************************)
Procedure writeheader;          {in Previewk}
begin
  with info do                  {set header values}
    begin
      nx      := points;
      nz      := aslen;
      w       := width;
      dt      := tsamp;
      v       := vel;
      sp      := calspace;
      Xavg    := navg;
      stn     := station;
      ln      := lenght;
      scns    := scans;
      mrk     := marker;
      zoffset :=depthoff;
      dcap    := datcap;
      srcfile := fin;
    end;
  blockwrite(fdata,info,(sizeof(info)+127) DIV 128);
end;    {writeheader}



(********************************** INITIALIZE *******************************)
(*                                                                          *)
(*                   set all variables to initial values                    *)
(*                                                                          *)
(****************************************************************************)
Procedure initialize;
var
  i: integer;

begin
  tsamp     := 0.267;
  vel       := 0.453;
  points    := 32;
  grlevels  := 4;
  marker    := 0;
  aslen     := 400;
  datcap    := 0;
  width     := 25;
  space     := 10/31;
  printscr  := false;
  calibrate := 'y';
  station   :=0;
  with info do
    begin
      nx    := 0;
      nz    := 0;
      w     := 0;
      dt    := 0.267;
      v     := 0.4e09;
      sp    := 0;
      Xavg  := 0;
      stn   := 0;
      ln    := 0;
      scns  := 0;
      mrk   := 0;
      zoffset := 0;
      dcap  := 0;
      srcfile := '';
    end;
end;   {initialize}


(***************************** PARMINPUT *************************************)
(*                                                                           *)
(*    read input and output file names as well as other parameters           *)
(*    All parameters are global variables; used for manual input only.       *)
(*                                                                           *)
(*****************************************************************************)
Procedure parminput;
BEGIN
  write( ' enter scource data root filename  ');
  readln(fin);
  fin := fin+'.dat';
  assign(datfile,fin);
  reset(datfile);
  write(  ' enter delimiter between A-scan traces  [0]  ');
  readln(marker);
  writeln(  ' enter data disk capacity [0 bytes] ');
  write(  ' A zero entry scales avialable data onto the whole screen ');
  readln(datcap);
  write( ' enter A-scan length [400] ');
  readln(aslen);
  write( ' enter approx. survey width [25ft] ');
  readln(width);
  write( ' enter A-scan spacing desired [10ft/31]  ');
  readln(space);
  lenght := longfilesize(datfile);
  if datcap = 0 then datcap := lenght;
```

```
    traces:=trunc((lenght/datcap)*(640/grlevels)); {640pixels,4 grey levels}
    scans := trunc(lenght/aslen);                  {traces = # of traces plotted}
                                                  {scans = total # of scans in data}
    avgr := (scans/width)/(1/space); {=(density of scans in data)/(desired
                                                density of scans in image}
    navg := trunc(avgr);
    if navg < 1 then navg := 1;      {does not allow scan average less than 1}
    avgplot := scans/traces;         {= #scans to average for each line plotted}
    writeln(' # of scans := ',scans,' of ',aslen:4,'each');
    writeln('navg = ',navg);
    writeln('avgplot ',avgplot);
    writeln('length := ',lenght:8:0);
    writeln('avgr', avgr);
    writeln('width', width);
    writeln('space', space);
    writeln('aslen', aslen);
    calspace := space;
    write(' Select plot intensity, H or [L] ');
    readln(intensity);
    if (intensity = 'H') or (intensity = 'h') then int_scale := 2;
    if (intensity = 'L') or (intensity = 'l')
      then int_scale := 1
      else int_scale := 1;
    writeln('Enter name for your output data file, ');
    write('ext.  .hdr  and  .prv  will be assigned  ');
    readln(fname);
    if fname<> ''then
      begin
        scrname := fname + '.prv';
        fout := concat(fname,'.hdr');
        assign(fdata,fout);
        rewrite(fdata);
        write('Enter data sampling period [.267 nS]          ---');
        write('Enter velocity [0.453 ft/nS]    ---');
        readln(vel);
        vel:=vel*1.0E9;
        write('enter depth offset (samples) --- ');
        readln(depthoff);
        write('Enter number of A-scans per data set [32] ---');
        readln(points);
      end;
    write(' Calibrate data width and station ? [Y] or N ');
    readln(calibrate);
    if (calibrate = 'n') or (calibrate = 'N') then calibrate := 'N'
       else calibrate := 'Y';
    write(' Print display ? Y or [N] ');
    readln(printprv);
    if (printprv ='y') or (printprv = 'Y') then printscr := true;

  END;    {parminput}
```

```
(****************************** PARMINPUT2 ************************************)
(*                                                                            *)
(*    read input and output file names as well as other parameters            *)
(*    All parameters are program global variables.  Modified for use          *)
(*    use with menu input program.                                            *)
(*                                                                            *)
(******************************************************************************)
Procedure parminput2;
BEGIN
  fin := fin;
  assign(datfile,fin);
  reset(datfile);
  space := calspace;
  lenght := longfilesize(datfile);
  if datcap < lenght then datcap := lenght;
  traces:=trunc((lenght/datcap)*(640/grlevels)); {640pixels,4 grey levels}
  scans := trunc(lenght/aslen);                  {traces = # of traces plotted}
                                    {scans = total # of scans in data}
  avgr := (scans/width)/(1/space); {=(density of scans in data)/(desired
                                                  density of scans in image}
  navg := trunc(avgr);
  if navg < 1 then navg := 1;       {does not allow scan average less than 1}
  avgplot := scans/traces;          {= #scans to average for each line plotted}
  calspace := space;
  if (intensity = 'H') or (intensity = 'h') then int_scale := 2;
  if (intensity = 'L') or (intensity = 'l')
    then int_scale := 1
    else int_scale := 1;
  if fname<> ''then                       {if output file name exist then}
    begin                                 {write header parameter into file}
      scrname := fname + '.prv';
      fout := concat(fname,'.hdr');
      assign(fdata,fout);
      rewrite(fdata);
    end;
  if (calibrate = 'n') or (calibrate = 'N') then calibrate := 'N'
     else calibrate := 'Y';
  if (printprv ='y') or (printprv = 'Y') then printscr := true;

END;    {parminput2}



(******************************** CALWIDTH ***********************************)
(*                                                                            *)
(*      calibrate the  displayed data width to standard units(feet)           *)
(*                                                                            *)
(******************************************************************************)
procedure calwidth;

var                                 {variables for data selection}
  cal_message:string[50];           {message}
  ch:char;                          {control character}
```

```
    st:string[6];                         {string buffer}
    CurrX,                                {present X coordinate value}
    CurrY:integer;                        {present Y coordinate value}
    IncrX,                                {X increment value}
    IncrY,                                {Y increment value}
    speed: integer;                       {scale factor for increased cursor motion}
    calib:boolean;                        {control for screen width calibration}
    Xavg,                                 {= navg}
    x1,x2:integer;                        {marked positions for calibration}
    dx:real;                              {scan to scan spacing}
    Scannum:integer;                      {the scan number at the cursor location}

  procedure Restoreback;                        {maintains the background}
  begin                                         {when called it recalls the original}
    selectscreen(2);                            {preview data screen and updates the}
    copyscreen;                                 {cursor location}
    selectscreen(1);
    Drawtextw(CurrX-7,CurrY,5,chr(27)+'5');{draw a diamond at new location}
    gotoxy(40,2);
    write(currx,'  ',curry);                    {writes diamond cursor position}
  end;    {Restoreback}


  procedure MoveUp(speed:integer);              { move polygon up }
  begin
    IncrY := 1*speed;
    CurrY:=CurrY+IncrY;
    Restoreback;                                {redraw background and cursor at}
    gotoxy(3,25);                               {new location}
    write(cal_message);
  end;    {MoveUp}

  procedure MoveDown(speed:integer);            { move polygon down }
  begin
    IncrY := 1*speed;
    CurrY:=CurrY-IncrY;
    Restoreback;                                {redraw background and cursor at}
    gotoxy(3,25);                               {new location}
    write(cal_message);
  end;    {Movedown}

  procedure MoveLeft(speed:integer);            { move polygon left }
  begin
    IncrX := 1*speed;
    CurrX:=CurrX-IncrX;                         { move to new position }
    Restoreback;                                {redraw background and cursor }
    ScanNum:=trunc(datcap/640/aslen*CurrX);{calculates present scan number}
    gotoxy(50,20);
    write('scan = ',ScanNum);
    gotoxy(3,25);
    write(cal_message);
  end;    {MoveLeft}

  procedure MoveRight(speed:integer);           { move polygon right }
```

```
begin
  IncrX :=1*speed;
  CurrX:=CurrX+IncrX;
  Restoreback;                          {redraw background and cursor }
  ScanNum:=trunc(datcap/640/aslen*CurrX);{calculates scan number}
  gotoxy(50,20);
  write('scan = ',ScanNum);
  gotoxy(3,25);
  write(cal_message);
end;    {MoveRight}



{main part of procedure Calwidth}
begin
  CopyScreen;                          {set initial values}
  CurrX:=50;
  CurrY:=200;
  IncrX:=0;
  Xavg := 8;
  calib:= true;
  restoreback;
  gotoxy(3,25);
  cal_message := 'move diamond to left mark,then depress  F1  ';
  write(cal_message);
  repeat
    gotoxy(3,25);
    write(cal_message);
    while keypressed do read(Kbd,Ch);           {read the keystroke}
    case ord(Ch) of
      72 : MoveUp(1);                           {up arrow?}
      75 : MoveLeft(1);                         {left arrow?}
      77 : MoveRight(1);                        {right arrow?}
      80 : MoveDown(1);                         {down arrow?}
      56 : MoveUp(15);                          {shift up arrow?}
      52 : MoveLeft(20);                        {shift left arrow?}
      54 : MoveRight(20);                       {shift right arrow?}
      50 : MoveDown(15);                        {shift down arrow?}
      59 : begin                                { F1 }
             cal_message := 'move diamond to right mark'+
                            ' then depress  F2  ';
             x1 := currX;
           end;
      60 : begin                                { F2 }
             cal_message := 'enter distance in feet between marks'+
                            '          ';
             gotoxy(3,25);
             write(cal_message);
             x2 := currX;
             gotoxy(45,25);
             readln(width);             {width between marks}
             dx := (width/abs(x2-x1))*aslen/datcap*640;
                        {dx = space = scan spacing = width/#scans
                         #scans = (% full display)*(total scans possible)
                         #scans = (x2-x1/640)*(datcap/aslen)}
            width := dx*scans;          {total data width}
            navg := trunc((scans/width)/(31/10)); {31 image trace, 10 ft wide}
                                     {   note navg is calculated such that
                                    reconstructed image will not exeed 10 ft }
            calspace:= dx*navg;         {spacing between averaged scans}
            gotoxy(50,25);
            write('calspace = ',calspace:5:3);
            cal_message:= 'move diamond to station marker'+
                          ' then press  F3  ';
            gotoxy(3,25); write(cal_message);
          end;
     61 : begin                                  { F3 }
            gotoxy(3,25);
            write('enter station [0 ft]'+
                  '                            ');
            gotoxy(27,25);
            readln(benchmark);             {bench mark is station number at}
            station := round(benchmark);   {annotated position}
            str(station,st);               {prepare annotation on screen}
            station := station - round(scannum*dx);
            selectscreen(2);                   {copies screen 2 to screen 1}
            setcolorblack;
            Drawtextw(CurrX-5,41,7,chr(27)+'4'); {draw the black diamond}
            for i := 0 to 8 do
              Drawtextw(CurrX-12+6*i,13,3,chr(27)+'4'); {blank 8 char space}
            setcolorwhite;                         {for station annotation}
            Drawtextw(CurrX-4,40,5,chr(27)+'5');     {draw the diamond}
            Drawtextw(currx-7,10,1,'sta='+st);       {write station}
            copyscreen;
            selectscreen(1);
            cal_message:= ' F4  completes calibration'+
                          ' or others to repeat ';
          end;
     62 : begin                                  { F4 }
          end;
   end;    {case}
   if ord(ch) <> 62 then ch := 'a';                   {62 is 'F4'}
  until ord(ch) = 62;    {repeat}
end;    {calwidth}


{********************************** PRINT_SCREEN ***************************}
{*                                                                         *}
(***************************************************************************)
procedure Print_Screen;
var
  device:FILE of integer;
  ffeed:integer;
  nz:integer;

begin
  inline   ($55/$cd/$05/$5d);        {print screen}
  assign(device,'prn');              {assign printer as device to be controlled}
  ffeed := 12;
  rewrite(device);
  write(device,ffeed);              {send formfeed control}
  close(device);
end;

(***************************** SET HEADER ****************************************)
(*                                                                                *)
(*      set global header parameters to the local input parameters values        *)
(*                                                                                *)
(**********************************************************************************)

PROCEDURE SET_HEADER;
begin
     with info do                   {set header values}
          begin
                 nx      := trunc(points);
                 nz      := trunc(aslen);
                 w       := width;
                 dt      := tsamp;
                 v       := vel;
                 sp      := calspace;
                 Xavg    := trunc(navg);
                 stn     := trunc(station);
                 ln      := lenght;
                 scns    := trunc(scans);
                 mrk     := trunc(marker);
                 zoffset := trunc(depthoff);
                 dcap    := datcap;
                 srcfile :=srcfile;
           END;
end; {set header variables}

(****************************** SET VAR  ****************************************)
(*                                                                                *)
(*  set program local variables to the global header parameter values            *)
(*                                                                                *)
(**********************************************************************************)

PROCEDURE SET_VAR;
    BEGIN
        with info do                   {set header values}
          begin
                 points := nx;
                 aslen := nz;
                 width := w;
                 tsamp := dt;
                 vel := v;
                 calspace := sp;
                 navg := Xavg;
                 station := stn;
                 lenght := ln;
                 scans := scns;
                 marker := mrk;
                 depthoff := zoffset;
                 datcap := dcap;
                { fin := srcfile;}
      end;
      intensity := char1;
      calibrate := char2;
      printprv  := char3;

 END; {SET_VAR}



(************************* MAIN PROCEDURE ************************************)
(*                                                                          *)
(****************************************************************************)
begin
  mark(heaptop);              {save the heap pointer for later memory release}

  grlevels := 4;              {moved here because of blocked out initialize}
  clrscr;
(*  initialize;               {set all variables to initial values}
  parminput; *)               {input parameters, used for mamual mode}
    which_screen := 2;        {2 = screen for preview}
    chain_return:=true;       {set return from workhorse flag}
    set_var;                  {set all the variables from global memory}
    parminput2;               {modified version of parameter input for menu}
  setup;                      {set up screen scales for drawing}
  repeat                      {seek for A-scan delimiter }
    read(datfile,a[1]);
  until a[1]=marker;
  markpos := filepos(datfile);{mark this first data byte position}
  gotoxy(68,25);
  write('DL =',lenght:7:0);
  for k := 1 to traces do     {read in A-scan traces }
    begin
      readpos:=(int((k-1)*avgplot)*aslen);
      longseek(datfile,readpos);   {position start read to first scan byte}
      gotoxy(60,25);
      write(readpos:7:0);
      for  I := 1 to aslen do  {read one scan}
        begin
          read(datfile,a[i]);
          b[i]:=a[i];
        end;
      plot;                   {plot one line}
    end; {traces}
  close(datfile);             {close raw input data file}
  beep(120);                 {makes a beep to signal display completion}
  if (calibrate = 'Y') or (calibrate = 'y') then
    CalWidth;                         {width calibration}
  annotate;
  if fname <> '' then                 {fname exist then write screen}
    begin
```

```
          savescreen(scrname);
          writeheader;
          close(fdata);              {close Tsaft data file}
        end;
      if printscr = true then
        begin
          swapscreen;
          Print_Screen;             {prints screen}
        end;
    { halt;  }
        leavegraphic;
        release(heaptop);           {give up heap space}
        clrscr;
        set_header;
        if auto_processing = 'Autoprocess'
        then begin          {execute autoprocessing, forward control to transfer}
               assign(workhorse,'transfwk.chn');
               chain(workhorse);
             end
        else begin          {return control to preview menu}
               assign(workhorse,'preview.chn');
               chain(workhorse);
             end;
    end. { main }
```

```
(*****************************  TRANSFER ****************************************)
(*                                                                              *)
(*   This program is for converting field data to T-saft input data.           *)
(*   In conjunction with Preview this program averages a number of             *)
(*   subsequent field radar scans to make a T-saft radar set of 63             *)
(*   A-scan traces which repersents a maximum of 10 survey linear              *)
(*   feet.                                                                      *)
(*   The outputs of the program are a user named data file with                *)
(*   a  .sft  and, a corresponding data plot file with a  .scr .  The          *)
(*   data file is compatible with T-saft and the screen file can be            *)
(*   used to perform the velocity determination.      5-13-87                  *)
(*                                                                              *)
(*   Further enhancements has been made to the program to perform              *)
(*   automatic processing of an entire disk of data.  Im this mode             *)
(*   output file names are also automatically indexed from 0 to 98.            *)
(*   The index takes precedence over the alpha characters in the file          *)
(*   name to form a maximum of 8 characters. NOTE: it is adviceable            *)
(*   to use alphabets only in the file name because T-saft would               *)
(*   interprete the numbers as an index.                                        *)
(*                                                    7-1-87                    *)
(********************************************************************************)

Program Transfer;

{ Global Required Definition of variables and constants }


    TYPE                                {writeheader variables}
         id = RECORD
                nx,                     {nx= # of A-scans in one data set}
                nz     :integer;        {nx = A-scan lenght, normally 400}
                w,                      {w = survey width}
                dt,                     {dt = sampling period}
                v      :real;           {v = radar propagation velecity}
                sp     :real;           {sp = averaged data scan-scan spacing}
                Xavg   :integer;        {Xavg = # of A-scan averaged per data A-scan}
                stn    :integer;        {station = data station}
                ln     :real;           {ln = data lenght}
                scns   :integer;        {scns =# of A-scansin the data}
                mrk    :integer;        {delimiter between A-scans in original data}
                zoffset:integer;        {template determined sample offset}
                dcap   :real;           { data capacity of a disk }
                srcfile:string[40];{scource data file}
                spare1:real;            {template determined velocity}
                spare2:integer;         {spare}
             END;
    VAR   { global variables }
         fname: string[40];             {name of I/O file}
         fin: string[40];               {name of input file}
         workhorse:file;                {chainfile assignee}
         fout:string[40];               {name of output file}
         which_screen:integer;          {which screen to use }
         chain_return:boolean;          {returning from chain program?}
```

```
        auto_processing:string[14];{automatic disk processing key}
        {the following characters are used for program flow control}
        char1,                   {plot intensity var in Preview}
        char2,                   {width calibtation var in Preview}
        char3,                   {print var in Preview}
        char4,                   {print var in Transfer}
        char5,                   {disk process var in Transfer}
        char6,                   {multiple processing var in T-saft}
        char7,                   {velocity stepping var in T-saft}
        char8,                   {print image var in T-saft}
        char9,                   {display format var in T-saft}
        char10,                {spare character variables}
        char11,                {spare character variables}
        char12,                {spare character variables}
        char13,                {spare character variables}
        char14,                {spare character variables}
        char15:char;           {spare character variables}
        sparename1,
        sparename2,
        sparename3,
        sparename4:string[40];   {spare strings}
        spareI1,
        spareI2,
        spareI3,
        spare14:integer;         {spare integers}
        spareR1,
        spareR2,
        spareR3,
        spareR4:real;            {spare reals}
        info:id;                 {header information}

{the above are global variables to be passed from progran to program during
 the chain calls. These variables are to be declared exactly as above in all
 programs wether used or not.  Tailoring will be done with the spares.}


TYPE
     Num = byte;
     string40 = string[40];
     heappointer =  integer;     {heap pointer}
CONST
     nxmax = 64;                 {max no of ascans per dataset}
     {aslen = 400; }             {no of data points per ascan}
     fullscaleX = 640;           {t-saft screen horizontal full scale}
     aslenmax  = 512;            {maximum A scan length}
     alen2     = 399;
     vertpixels= 180;            {vertical pixels}
VAR
     {fname:string[25];}         {name of I/O file}
     {fin: string[25];}          {name of input file}
     {fout:string[40];}          {output data file name}
     parsename:string[40];       {output data file name storage}
     A: array[1..400] of Num;    {byte input data buffer}
     b: array[1..400] of integer; {integer plot data buffer}
```

```
        buff: array[0..127] of num;  {input parameter buffer}
        fdata: FILE;                 {output disk file}
        datfile: file of num;        {raw data disk file}
        print,                       {control for data plot printing}
        printrawdata: char;          {control for printing display}
        aslen,                       {length of radar traces, normally 400}
        n,K,I,j,l,m,                 {general indices}
        traces,                      {# lines in data plot}
        delimiter,                   {delimiter between A-scan traces}
        navg,                        {# radar scans to average for data set}
        xpos: integer;               {screen x coordinate}
        points,                      {number of A-scans per data set}
        grlevels,                    {# of grey levels in plot}
        markpos,                     {position of first data point}
        station,                     {ground station reference}
        stationx,                    {cursor ground station}
        depthoff,                    {depth offset}
        scans,                       {number of scans}
        datarange: integer;          {range of data in data set}
        traceavg,                    {average value of a scan; for DC removal}
        avgplot : real;              {#data scans to average for one plot line}
        datcap,                      {data disk capacity}
        avgr,                        {#data scan to average for data set}
        space,                       {A-scan spacing}
        spare,                       {spare real variable on header}
        width,                       {approx. survey width}
        lenght,                      {length in bytes of radar data file}
        readpos,                     {where to read data for plotting}
        calspace,                    {calculated scan spacing for saft data file}
        startscan,                   {first scan in the reconstructed image}
        datapos: real;               {position of data in scource file}
        printscr,                    {control variable for plot printing}
        fill_lead_zero,              {leading zero control when there is no data}
        fill_trail_zero: boolean;    {trailing zero control when               }
        scannum: integer;            {cursor scan number position}
        OK:boolean;                  {file existence flag}
        process:char;                {automatic data transfer}
        vel,                         {velocity}
        tsamp:real;                  {data sampling period}
        heaptop:heappointer;         {heap pointer}

   var                               {variables for data selection}
     cal_message:string[66];         {message}
     ch:char;                        {control character}
     st:string[6];                   {string buffer}
     CurrX,                          {present X coordinate value}
     CurrY:integer;                  {present Y coordinate value}
     IncrX,                          {X increment value}
     IncrY,                          {Y increment value}
     speed: integer;                 {scale factor for increased cursor motion}
     calib:boolean;                  {control for screen width calibration}
     Xavg,                           {= navg}
     x1,x2:integer;                  {marked positions for calibration}
     dx:real;                        {scan to scan spacing}
```

```
  index:integer;                    {output file name index for auto process}

(***************** end of variable declaration section *******************)

{$R+}
{$V-}

{These include files are defined in the Borland Turbo Graphix Toolbox
 manual and must be included in this order.  They are used in this program
 by procedure calls coded in the program.  For further information consult
 the Borland Turbo Graphix Toolbox manual.}
  {$I c: graphix typedef.sys}                      {variable declaration}
  {$I c: graphix graphix.sys}                      {definitions and routines}
  {$I c: graphix kernel.sys}                       {support routine}
  {$I c: graphix windows.sys}                      {graphics window manipulation

(*************************** TIME and DATE *******************************)
(*                                                                        *)
(*                            time and date                               *)
(*                                                                        *)
(**************************************************************************)
procedure timedate(var hour,min,month,day,year:integer);
type
  registers = record
                ax,bx,cx,dx,bp,si,di,ds,es,flags:integer;
              end;
var
  regs: registers;

begin
  with regs do            {time of day}
    begin
      ax := $2C00;
      msdos(regs);        {MS DOS function call to DOS time function}
      hour := hi(cx);
      min := lo(cx);
    end;
  with regs do            {date}
    begin
      ax := $2A00;
      msdos(regs);        {MS DOS function call to DOS date function}
      year := cx;
      month := hi(dx);
      day := lo(dx);
    end;
end;    {timedate}

{******************************  PRINT_SCREEN  ***************************}
(*                                                                        *)
(*                                                                        *)
(**************************************************************************)
procedure Print_Screen;
var
  device: FILE of integer;
  ffeed: integer;                    {code for form feed}

begin
  inline   ($55/$cd/$05/$5d);        {print screen}
  assign(device,'prn');              {assign control destination to printer}
  ffeed := 12;
  rewrite(device);
  write(device,ffeed);               {send form feed control}
  close(device);
end;    {print_screen}

(************************ SET HEADER ************************************)
(*                                                                      *)
(*     set global header parameters to the local input parameters values *)
(*                                                                      *)
(************************************************************************)

PROCEDURE SET_HEADER;
begin
     with info do                    {set header values}
          begin
                 nx     := trunc(points);
                 nz     := trunc(aslen);
                 w      := width;
                 dt     := tsamp;
                 v      := vel;
                 sp     := calspace;
                 Xavg   := trunc(navg);
                 stn    := trunc(station);
                 ln     := lenght;
                 scns   := trunc(scans);
                 mrk    := trunc(delimiter);
                 zoffset := trunc(depthoff);
                 dcap   := datcap;
                 srcfile :=srcfile;
           END;
        sparename3 := fout;
end; {set header variables}

(**************************** SET VAR  **********************************)
(*                                                                      *)
(*  set program local variables to the global header parameter values   *)
(*                                                                      *)
(************************************************************************)

PROCEDURE SET_VAR;
    BEGIN
        with info do                    {set header values}
          begin
                 points := nx;
                 aslen := nz;
                 width := w;
                 tsamp := dt;
                 vel := v;
```

```
                      calspace := sp;
                      navg := Xavg;
                      station := stn;
                      lenght := ln;
                      scans := scns;
                      delimiter := mrk;
                      depthoff := zoffset;
                      datcap := dcap;
                      fin := srcfile;
             end;
             printrawdata := char4;
             process      := char5;

         END; {SET_VAR}


(********************************* PARSE ****************************************)
(*                                                                              *)
(*      Procedure parses the name in fname into its path, rootname,index        *)
(*      and extension.  The index is appended by the procedure and              *)
(*      the fname is reassenbled with this new index.  The index will occupy    *)
(*      no more than the two last position of the eight character name.         *)
(*      This allow a miminum of six character for the name and a maximum        *)
(*      of 98 for the index value.                                              *)
(*                                                                              *)
(*      NOTE: File names are restricted to valid non numeric characters         *)
(*      only. Any number in the name will be interpreted as the index           *)
(*      of the name and be used to generate the resualtant name.                *)
(*                                                                              *)
(********************************************************************************)

procedure parse(var name:string40; var index:integer);
var
  fpath,                    {path name}
  faux1,                    {root file name}
  faux2:string[40];         {intermediate path name segment}
  indst:string[2];          {string equvalent of index}
begin
  fpath := '';
  faux1 := '';
  faux2 := '';
  indst := '';
  faux2 := copy(name,1,pos(':',name));
  fpath := concat(fpath,faux2);
  delete(name,1,pos(':',name));
writeln('name after drive deleted ',name);
repeat
  faux2 := copy(name,1,pos(' ',name));        {identify directory levels}
  fpath := concat(fpath,faux2);               {add next path segment to path name}
  delete(name,1,pos(' ',name));               {remove path segment}
until length(faux2) = 0;                      {no more path name segment}
writeln('fpath=',fpath);
  str(index,indst);                           {set indst to index value}
  if pos('.',name) <> 0
  then  faux1 := copy(name,1,pos('.',name)-1)
  else  faux1 := name;
  name := copy(faux1,1,8-length(indst));
writeln('root name =',name);
  name := concat(fpath,name,indst);          {name is made up of a) the path }
writeln('final output file name= ',name);
end;                                         {b) the root name c)the index and}
                                             {d) the '.sft' appended in pipedata}


(***************************** PROCEDURE ANNOTATE ***************************)
(*                                                                          *)
(*                 write pertinent parameters to the screen                 *)
(*                                                                          *)
(****************************************************************************)
procedure annotate;
var
  hour,
  min,
  month,
  day,
  year:integer;

begin
  timedate(hour,min,month,day,year);
  gotoxy(1,25); write('                                                       '+
                      '                             ');
  gotoxy(2,25); write(fout);
  gotoxy(15,25); write('X=',navg);
  gotoxy(20,25); write('W =',width:4:1);
  gotoxy(30,25); write('Sp=',Calspace:5:3);
  gotoxy(40,25); write('T=',tsamp*1E09:5:3);
  gotoxy(50,25); write(hour:2,':',min:2,
                      ' ',month:2,'-',day:2,'-',year:2);
  gotoxy(68,25); write('DL=',lenght:6:0);
  copyscreen;
end;    {annotate}


(******************************* PROCEDURE PLOT ****************************)
(*                                                                          *)
(*    plot data scans on screen                                             *)
(*    In this graphics version y=0 is at the bottom                         *)
(*                                                                          *)
(****************************************************************************)
Procedure plot;
begin
  xpos :=round(k*grlevels*(calspace*(2*points-1)/20));
  FOR i:=1 TO aslen DO
    begin
      n:= round((b[i])*grlevels/128);
      if n >grlevels then n:=grlevels;           {upper limit}
      if n <-grlevels then n:= -grlevels;        {lower limit}
      {n := round((b[i]-127)/(128/grlevels));}
      drawpoint(xpos,aslen-i);                    {draw reference line}
      drawline(xpos,aslen-i,xpos+n,aslen-i);      {draw data}
    end;  {FOR}
end;    {plot}


(**************************** PROCEDURE WRITEDATA ***************************)
(*                                                                          *)
(*       writes buffer; first 400 points is data, the rest is 0             *)
(*                                                                          *)
(****************************************************************************)
Procedure writedata;

VAR
  temp: integer;
  xbuf:ARRAY[1..aslenmax] OF integer;
  i: integer;

BEGIN
  for i := 1 to aslenmax do xbuf[i] := 0;  {clear buffer}
  FOR i:=1 TO aslen DO
    BEGIN
      xbuf[i]:=b[i];
    END;
  blockwrite(fdata,xbuf,(2*aslen+127) DIV 128);
END;  {writedata}


(************************* PROCEDURE WRITEHEADER ******************************)
(*                                                                            *)
(*       write pertinent data parameters in data file header                  *)
(*                                                                            *)
(******************************************************************************)
Procedure writeheader;
begin
  width := calspace*(points-1);
  tsamp:=tsamp;
  with info do                          {set header values}
    begin
      nx := points;
      nz := aslen;
      w := width;
      dt := tsamp;
      v := vel;
      sp := calspace;
      Xavg := navg;
      stn := stationx;
      ln  := lenght;
      scns := scans;
      mrk := delimiter;
      zoffset := depthoff;
      dcap := datcap;
      srcfile :=srcfile;
    end;
  blockwrite(fdata,info,(sizeof(info)+127) DIV 128);
end;    {writeheader}


(**************************** PROCEDURE SETUP *******************************)
(*                                                                          *)
(*        defines the display areas for input data drawing                  *)
(*                                                                          *)
(****************************************************************************)
Procedure setup;
var
  st:string[6];

begin
  clrscr;
  EnterGraphic;                                 {set graphic mode}
  DefineWindow(1,0,0,79,180);                   {define drawing and annotation}
  Defineworld(1,0,0,639,aslen);                 {areas}
  DefineWindow(2,0,180,79,199);
  Defineworld(2,0,0,639,20);
  SelectWorld(2);
  SelectWindow(2);                              {draw horizontal scales in}
  for i := 1 to 10 do                           {window #2}
    begin
      drawline((i-1)*64,17,(i-1)*64,20);        {draw 10 tick marks along bottom}
      drawtextW(0,13,1,'0');
      str(i*datcap*0.1/aslen:4:0,st);
      drawtextW(i*64-15,13,1,st);
      drawtextW(610,13,1,'scans');
    end;
  SelectWorld(1);                               {return to data drawing area}
  SelectWindow(1);
  SetBackground(0);
  Drawborder;
  setcolorwhite;
end;     {setup}



(***************************** SETUP 2 **************************************)
(*                                                                          *)
(*             defines screen area for output data drawing                  *)
(*                                                                          *)
(****************************************************************************)
Procedure setup2;
var
  st: string[3];                           {station numbers}
  stnoff,                                  {station offset}
  scale:integer;
```

```
BEGIN
  scale := 20;
  clrscr;
  EnterGraphic;
  DefineWindow(1,6,0,79,180);           {data drawing window}
  Defineworld(1,0,0,756,399);
  DefineWindow(2,0,0,5,190);            {Vertical scale window}
  Defineworld(2,0,0,48,190);
  DefineWindow(3,0,181,79,199);         {Hz scale window}
  Defineworld(3,0,0,639,19);
  Selectworld(2);                       {select vert scale drawing area}
  SelectWindow(2);
  for i := 0 to scale do                {make and label scale marks}
    begin
      drawline(44,i*vertpixels/scale+10,48,i*vertpixels/scale+10);
      str(round(i*aslen/scale):3,st);
      if (not odd(i)) and (i > 0) then
        Drawtext(25,round(i*vertpixels/scale),1,st);
    end;
  Drawtext(25,2,1,' 0');
  Drawtext(0,64,1,'S');                         {label vert axis}
  Drawtext(0,72,1,'a');
  Drawtext(0,80,1,'m');
  Drawtext(0,88,1,'p');
  Drawtext(0,96,1,'l');
  Drawtext(0,104,1,'e');
  Drawtext(0,112,1,'s');
  selectWorld(3);                               {select Hz scale display area}
  selectwindow(3);
  stnoff := abs(station + round((datapos/aslen)*(calspace/navg))
                         mod 100); {calculate station # of first data scan}
  for i:= 0 to 10 do
    begin
      drawline(i*(fullscaleX-50)/10+48,15,i*(fullscaleX-50)/10+48,19);
      if station >= 0                           {draw bottom scale}
        then str(round(2*i+stnoff):3,st)        {write increasing scale}
        else str(round(stnoff-2*i):3,st);       {write decreasing scale}
      DrawtextW(i*(fullscaleX-50)/10+43,12,1,st);
    end;    {for}
      DrawtextW(i*(fullscaleX-50)/10+33,12,1,'   ');  {blank out last scale entry}
      DrawtextW(i*(fullscaleX-50)/10+33,12,1,st);     {rewrite last scale entry}
  DrawtextW(300,5,1,'Station (ft)');
  DrawtextW(0,12,1,'sta ');
  str(abs(station div 100):2,st);               {label Hz axis}
  DrawtextW(20,12,1,st);
  DrawtextW(32,12,1,'+');
  SelectWorld(1);                               {select data drawing area}
  SelectWindow(1);
  SetBackground(0);
  Drawborder;
END;    {setup}
```

```
(************************** PIPE DATA ****************************************)
(*                                                                           *)
(*   Reads data from field data file, rewrites it into integer format       *)
(*   for T-saft and call plot routine.  Also averages succeeding scans      *)
(*   and removes DC level                                                   *)
(*                                                                           *)
(*****************************************************************************)
procedure pipedata;
begin
  assign(datfile,fin);                    {fin = raw input data file}
  {$I-}  reset(datfile) {$I+};            {open input raw data file}
  OK := (IOresult = 0);
  if not OK then
              begin
                writeln('Cannot find data file ',fin);
                write(' Quit '); halt;                    {quit program}
              end;
  assign(fdata,fout+'.sft');              {open output T-saft data file}
  rewrite(fdata);
  stationx:= station + round((datapos/aslen+navg*(points-1)/2)*calspace/navg);
  writeheader;                            {write header into t-saft data file}
  clrscr;
  setup2;                                 {set screen for ouput data drawing}
  if not ((process = 'y') or (process = 'Y'))
  then begin
        datapos:= currX*datcap/fullscaleX-navg*(points-1)*aslen;
                                           {calculate the first data}
         if abs(datapos)<2*aslen then datapos := 0; {position to read from the}
       end;
  if datapos < 0                                 {input data; if pos is close}
    then                                         {to 0 then snap to 0}
      fill_lead_zero := true
    else                 {advance read pointer to first A-scan of interest}
      longseek(datfile,datapos);
  repeat                                 {then seek for A-scan delimiter}
    read(datfile,a[1]);
  until a[1]=delimiter;
  for k := 1 to 2*points-1 do            {read in navg*traces A-scan traces }
    begin
      for  I := 1 to aslen do b[i]:=0;   {zero array b}
      for j := 1 to navg do              {averages navg # of A-scans}
        begin
          for  I := 1 to aslen do
            begin
              if (not fill_lead_zero) and (not fill_trail_zero)
              then
                begin
                  if EOF(datfile)
                  then
                    begin
                      {gotoxy(10,10); write('  End of data reached  ');}
                      fill_trail_zero:= true;
                    end
                  else
```

```
                    begin
                      read(datfile,A[i]);    {finally; reads one data point}
                      b[i]:=b[i] + round(a[i]/navg);
                    end;
                end
              else b[i] := 0;      {out of data area, set b = 0}
            end;  {for I}
        end;   {for j}
      datapos := datapos + aslen*navg;   {moves data positon pointer forward}
      gotoxy(0,25); write('  data position = ',datapos:7:0,'  ');
      if datapos >= 0 then fill_lead_zero := false;
      if k < (2*points) then            {write 2*point-1 averaged A-scans}
        begin
          traceavg := 0;           {remove DC bias in each averaged A-scan by
                                    subtracting each point with the average value
                                    of each trace}
          for I := 1 to aslen do traceavg := traceavg + b[i];
          traceavg := traceavg/aslen;
          for I := 1 to aslen do b[i] := b[i] - round(traceavg);
          plot;                          {plot one trace}
          writedata;                     {write one data trace into output file}
        end;
    end; {for k}
  if (process = 'Y') or (process = 'y') then
  datapos := datapos-aslen*(points-1)*navg-10;{backup half data width for next}
  annotate;                              {write pertinent info on screen}
  savescreen(fout+'.scr');
  close(datfile);
  close(fdata);
  if (printrawdata = 'Y') or (printrawdata = 'y')  {print saft data screen}
    then Print_Screen;
end;    {pipedata}



(************** procedures for select data on screen *********************)
(*                                                                      *)
(************************************************************************)

procedure Restoreback;                {maintains the background}
begin
  selectscreen(2);                    {copies screen 2 to screen 1}
  copyscreen;
  selectscreen(1);
  Drawtextw(CurrX-7,CurrY,5,chr(27)+'5');
  drawline(CurrX-datarange/2,curry-5,currx+datarange/2,curry-5);
  drawline(CurrX-datarange/4,curry-2,currx+datarange/4,curry-2);
  datapos:= currX*datcap/fullscaleX-navg*(points-1)*aslen;
  ScanNum:=round((datapos/aslen)+navg*(points-1));
  gotoxy(56,20);                                {calculated .alue of scannum}
  write('    ');          {note: scannum is actual data s an number at diamond}
  gotoxy(50,20);
  write('scan = ',ScanNum);
  gotoxy(40,2);
  write(currx,'  ',curry);
  stationx:= station + round((datapos/aslen+navg*(points-1)/2)*calspace/navg);
                       {calculate new station number for T-saft data file}
  gotoxy(59,21);
  write('     ');
  gotoxy(50,21);
  write('station = ',stationx);
end;    {Restoreback}


procedure MoveUp(speed:integer);      {move polygon up}
begin
  IncrY := 1*speed;
  CurrY:=CurrY+IncrY;
  Restoreback;
  gotoxy(3,24);
  write(cal_message);
end;    {MoveUp}

procedure MoveDown(speed:integer);    {move polygon down}
begin
  IncrY := 1*speed;
  CurrY:=CurrY-IncrY;
  Restoreback;
  gotoxy(3,24);
  write(cal_message);
end;    {MoveDown}

procedure MoveLeft(speed:integer);            {move polygon left}
begin
  IncrX := 1*speed;
  CurrX := CurrX-IncrX;
  Restoreback;
  gotoxy(3,24);
  write(cal_message);
end;    {moveLeft}


procedure MoveRight(speed:integer);           {move polygon right}
begin
  IncrX :=1*speed;
  CurrX:=CurrX+IncrX;
  Restoreback;
  gotoxy(3,24);
  write(cal_message);
end;    {MoveRight}


(****************************** PARMINPUT **********************************)
(*                                                                         *)
(*          parameter input; used in manual data input                     *)
(*                                                                         *)
```

```
(**************************************************************************)
Procedure parminput;
BEGIN
repeat
  write( ' enter scource data root filename  ');
  readln(fname);
  assign(fdata,fname+'.hdr');
  {$I-} reset(fdata) {$I+};
  OK := (IOresult = 0);
  if not OK then writeln('Cannot find file ',fname);
until OK;
  write('enter ouput file name                                   ');
  readln(fout);
  write('Print raw data ? Y or [N]                               ');
  readln(printrawdata);
  process := 'N';
  write('process entire disk? Y or [N] ');
  readln(process);
  write('place data disk in drive A, then press enter       ');
  readln;
  blockread(fdata,buff,1);               {read header information}
  move(buff,info,sizeof(info));
  with info do                           {set header values}
    begin
      points := nx;
      aslen := nz;
      width := w;
      tsamp := dt;
      vel := v;
      calspace := sp;
      navg := Xavg;
      station := stn;
      lenght := ln;
      scans := scns;
      delimiter := mrk;
      depthoff := zoffset;
      datcap := dcap;
      fin := srcfile;
      close(fdata);
    END;    {with}
END;   {parminput}


(************************ LIST VARIABLES ******************************)
(*                                                                      *)
(**************************************************************************)

procedure list_Var;
  Begin
      writeln;
      writeln('  points   := ',points:8);
      writeln('  aslen    := ',aslen:8);
      writeln('  width    := ',width:8:3);
      writeln('  tsamp    := ',tsamp*1e9:8:4);
      writeln('  vel      := ',vel*1e-9:8:4);
      writeln('  calspace := ',calspace:8:4);
      writeln('  navg     := ',navg:8);
      writeln('  station  := ',station:8);
      writeln('  lenght   := ',lenght:8:0);
      writeln('  scans    := ', scans:8);
      writeln('  delimiter:= ',delimiter:8);
      writeln('  depth offset:= ',depthoff);
      writeln('  datcap   := ',datcap:8:0);
      writeln('  fname    := ',fname);
      writeln('  fin      := ',fin);
      writeln('  fout     := ',fout);
      writeln('  print    := ',printrawdata);
      writeln('  process  := ',process);
      writeln(' enter CR to proceed ');
      readln;
   End;


(************************* INITIALIZATION **********************************)
(*                                                                         *)
(***************************************************************************)
Procedure initialize;
var
  i: integer;

begin
  index := 0;
  grlevels := 12;
  printscr:=false;
  fill_trail_zero:=false;
  fill_lead_zero:=false;
  Ch := 'a';
end;    {initialize}


(************************* MAIN PROCEDURE **********************************)
(*                                                                         *)
(***************************************************************************)


begin
  mark(heaptop);              {save heap pointer for later memory release}
  InitGraphic;                {master initialize of the graphic procedures}
  which_screen := 2;          {2 = screen for transfer}
  chain_return:=true;         {set return from workhorse flag}
  initialize;                 {set all variables to initial values}
  {parminput;}                {all parameters are global variables}
  set_var;                    {proc sets local var to globals passed from menu}
  {list_var;}                 {activate for parameter transfer checking}
  clrscr;
  if auto_processing = 'Autoprocess'  {set output name to same as input name}
```

```
      then fout := fname;                {when in auto processing mode}
      parsename := fout;

    {main body of procedure selectdata}
      CopyScreen;
      dataRange := round(aslen/datcap*2.0*(points-1)*navg*fullscaleX);
      datapos:=0;                         {first data position to transfer}
      if (process = 'Y')or(process = 'y')  {image starts at begining if auto proc}
             then datapos := -(points-1)*aslen;
      currX:= round(fullscaleX/datcap*(datapos+aslen*(points-1)*navg));
      CurrY:=200;
      IncrX:=0;

    IF (process='Y') or (process='y')   {automatic disk processing}
     THEN
        begin
            l:=0;
          repeat    {write Tsaft data files until end of data or 99 files written}
            begin
              index:=1;
              fout := parsename;
              parse(fout,index);          {generate next output file name}
              currX:= round(fullscaleX/datcap*(datapos+aslen*(points-1)*navg));
                                   { move cursor forward to next data area}
              setup;                                {setup screen for reloading}
              loadscreen(fname+'.prv');                {reload preview screen}
              copyscreen;
              restoreback;
              gotoxy(0,25); write('  data position = ',datapos:7:0,'  ');
              delay(3000);
              pipedata;               {proceed to write Tsaft data file}
              l := l+1;
            end;
          until fill_trail_zero or (l > 98); {stop when E 0 data or index exceeded}
          writeln('Normal exit, last file processed ',fout,'               ');
        end                             {end of automatic processing}


     ELSE              {**  start manual data selection  **}
        begin
  loadscreen(fname+'.prv');
      copyscreen;
      restoreback;
      gotoxy(3,24);
      cal_message
         := 'Center diamond in data area, Press  F9  to write,  F10  to quit    ';
      write(cal_message);
      repeat
        gotoxy(1,24);
        write(cal_message);
        while keypressed do read(Kbd,Ch);             {read the keystroke}
        case ord(Ch) of
          72 : MoveUp(1);                             {up arrow?}
          75 : MoveLeft(1);                              {left arrow?}
          77 : MoveRight(1);                             {right arrow?}
          80 : MoveDown(1);                              {down arrow?}
          56 : MoveUp(15);                               {shift up arrow?}
          52 : MoveLeft(20);                             {shift left arrow?}
          54 : MoveRight(20);                            {shift right arrow?}
          50 : MoveDown(15);                             {shift down arrow?}
    {F8}  66 : begin
                 fill_trail_zero:=false;                 {reset flag}
                 gotoxy(1,25);
                 write('                                                        ');
                 gotoxy(2,25);
                 write('enter output filename, .sft appended  ');
                 readln(fout);
                 setup;                                  {setup screen for reloading}
                 loadscreen(fname+'.prv');                  {reload preview screen}
                 copyscreen;
                 restoreback;                            {restore cursor}
                 gotoxy(1,24);
               cal_message
           := 'Center diamond in data area, Press  F9  to write,  F10  to quit    ';
                 write(cal_message);
               end;
    {F9}  67 : begin
                 pipedata;
                  cal_message
               := 'Press  F10  to quit,  F8  to write another T-saft data set      ';
               end;
        end;
    {F10}
        if ord(ch) <> 68 then ch := 'a';
      until ord(ch) = 68;
    end;    {** end manual data selection **}

        leavegraphic;                     {go back to text mode}
        clrscr;
        set_header;
      {  halt;}
        release(heaptop);                 {give up heap space}
        if auto_processing = 'Autoprocess'{if auto processing then transfer}
        then                                 {control to T-saft workhorse}
          begin
            index := 0;
            fname:=parsename;              {fname set to Ist data file name}
            parse(fname,index);
            fname:=fname+'.sft';
            assign(workhorse,'tsaftwk.chn');
            chain(workhorse);                  {transfer control to tsaftwk}
          end
        else
          begin                              {go back to Transfer menu not not}
            assign(workhorse,'transfer.chn');      {auto processing}
            chain(workhorse);
          end;
```

```
end.      {Transfer}
```

```
(********************************** TSAFT ************************************)
(*                                                                           *)
{* This is the time domain synthetic aperture focusing technique (T-SAFT)    *}
(* program. It performs a two dimensional reconstruction of the vertical     *)
(* ground profile from a consecutive series of digitized radar pulse         *)
(* reflections.  In a nutshell, the image reconstruction algorithm           *)
(* extracts the signal information that pertains to a particular location   *)
(* in the reconstruction plane from the multiple radar reflections and       *)
(* places the resultant at that location.  This process is repeated for      *)
(* each image pixel to obtain a complete 2D reconstruction.                  *)
(*    T-saft takes in a data set that is double wide (63x400) and produces   *)
(* a complete reconstruction image of the middle half (32x400).  Thus,       *)
(* data for tsaft should overlap by half the image width.                    *)
(*                                                                           *)
(* This T-saft program has several features:                                 *)
(*      1) built in stationary target remover.                               *)
(*      2) allows depth offset as a parameter.                               *)
(*      3) dual depth scale reconstruction, 10 or [20]ft.                    *)
(*      4) horizontal distances are automatically scaled                     *)
(*         to a fixed 10 ft full scale using scan spacing information.       *)
(*      5) filled in wiggle display,wiggle to right is positive.             *)
(*      6) output screen is automatically annotated with pertinent data.     *)
(*      7) Automatic printing of display and formfeed.                       *)
(*      8) Velocity steping in 5 steps centered around the input velocity,   *)
(*         each step is 5% of entered velocity.                              *)
(*                                                                           *)
(*                                                      7-1-87               *)
(*****************************************************************************)

PROGRAM T_SAFT;

{ Global Required Definition of variables and constants }


    TYPE                                {writeheader variables}
          id = RECORD
                 nx,                    {nx= # of A-scans in one data set}
                 nz     :integer;       {nx = A-scan lenght, normally 400}
                 w,                     {w = survey width}
                 dt,                    {dt = sampling period}
                 v      :real;          {v = radar propagation velecity}
                 sp     :real;          {sp = averaged data scan-scan spacing}
                 Xavg   :integer;       {Xavg = # of A-scan averaged per data A-scan}
                 stn    :integer;       {station = data station}
                 ln     :real;          {ln = data lenght}
                 scns   :integer;       {scns =# of A-scansin the data}
                 mrk    :integer;       {delimiter between A-scans in original data}
                 zoffset:integer;       {template determined sample offset}
                 dcap   :real;          { data capacity of a disk }
                 srcfile:string[40];{scource data file}
                 spare1:real;           {template determined velocity}
                 spare2:integer;        {spare}
```

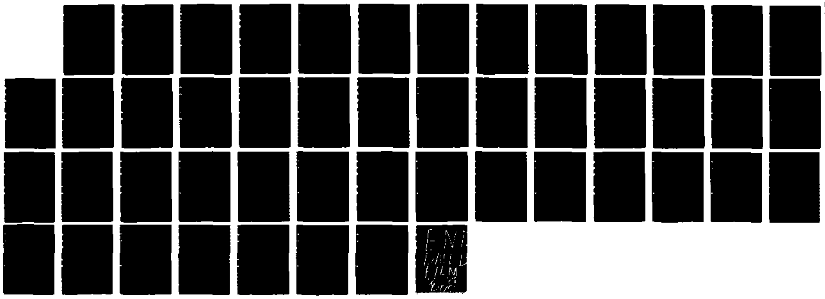
```
              END;
     VAR    { global variables }
            fname: string[40];          {name of I/O file}
            fin: string[40];            {name of input file}
            workhorse:file;             {chainfile assignee}
            scrname:string[40];         {name of output file}
            which_screen:integer;       {which screen to use }
            chain_return:boolean;       {returning from chain program?}
            auto_processing:string[14];{automatic disk processing key}
            {the following characters are used for program flow control}
            char1,                      {plot intensity var in Preview}
            char2,                      {width calibtation var in Preview}
            char3,                      {print var in Preview}
            char4,                      {print var in Transfer}
            char5,                      {disk process var in Transfer}
            char6,                      {multiple processing var in T-saft}
            char7,                      {velocity stepping var in T-saft}
            char8,                      {print image var in T-saft}
            char9,                      {display format var in T-saft}
            char10,                 {spare character variables}
            char11,                 {spare character variables}
            char12,                 {spare character variables}
            char13,                 {spare character variables}
            char14,                 {spare character variables}
            char15:char;            {spare character variables}
            sparename1,
            sparename2,
            sparename3,
            sparename4:string[40];      {spare strings}
            spareI1,
            spareI2,
            spareI3,
            spare14:integer;            {spare integers}
            spareR1,
            spareR2,
            spareR3,
            spareR4:real;               {spare reals}
            info:id;                    {header information}

{the above are global variables to be passed from progran to program during
 the chain calls. These variables are to be declared exactly as above in all
 programs wether used or not.  Tailoring will be done with the spares.}


CONST
  maxcolor=12;              {number of display colors}
  aslen=400;                {A-scan data lenght}
  alen2=399;                {aslen - 1}
  aslenmax=512;             {maximum A-scan lenght}
  dmax=400;                 {depth dimension of image array}
  nxmax=128;                {max number of A-scans}
  convlen=255;              { = 2*nxmax-1;}
  ntick=4;
  pltx=256;
  plty=200;
  maxshadecolor=3;          {number of display color}
  vertpixels=180;           {vertical display pixels}

TYPE

  pas    =  arr_a;                          {pointer to array a}
  arr_a  = ARRAY[0..aslen] OF integer;
  arr_s  = ARRAY[0..nxmax] OF pas;          {array of pointers to sensor data}
  pia    =  arr_ia;                         {pointer to array ai}
  arr_ia = ARRAY[0..dmax] OF real;
  arr_ix = ARRAY[0..nxmax] OF pia;          {image array type}
  arr_p  = ARRAY[0..convlen] OF real;       {reconstructed image array by  fold }
  arr_r  = ARRAY[0..nxmax] OF integer;      {hyperbola generated by  delay }

VAR
{  fname: string[40];  }          {file name of T-saft input data file}
{  fin :string[14];    }          {reserved input file name}
  data: FILE;                   { internal input data file name}
{  info:id;            }          {header parameters}
  buff: ARRAY[0..127] OF byte;   {input buffer for blocking data and header}
  ibuf: ARRAY[0..aslenmax] OF integer; {input data buffer}
  avg: array[0..aslen] of real;  {average trace of input scans}
  space,                    {scan spacing in T-saft input data}
  vt,                       {vel x tsamp, vertical distance per sample}
  tsamp,                    {digitization sampling period}
  vel,                      {radar propagation velocity}
  vel0,                     { default or assigned initial velocity}
  mag: real;                {magnitude to plot on the image plot}
  imax,                     {maximum image intensity}
  imin: real;               {minimum                 }
  image: arr_ix;            {reconstructed output image array}
  sensor: arr_s;            {raw data input array}
  smax,                     {maximun sensor value}
  xsize,                    {x dimension of image (same as points and asize)}
  zsize,                    {z dimension of image (same as aslen)}
  color,                    {number of color to plot, maximum is 4}
  x,y,i,j,iterate: integer;         {general indicies}
  step: char;               {velocity stepping control}
  stepping:boolean;         {                        }
  print:char;               {output print control}
  printscreen: boolean;       {output image screen printing control}
  depthoff:integer;         {depth offset of raw data in samples;
                                 +ve means top ground is delay x samples}
  station,                  {ground station identification}
  scale: integer;           {reconstruction scale 10 or [20] ft}
  aslenv,                   {same as aslen}
  navg,                     {# of raw A-scans averaged to amke one T-saft scan}
  scans,                    {# of scans in raw survey data}
  delimiter,                {scan delimiter in raw survey data}
  grlevels:integer;         {# of grey levels in the display}
  width,                    {width of reconstructed image}
  calspace,                 {scan spacing in reconstucted image same as space}
  datcap,                   {data capacity of raw survey data disk (bytes)}
  lenght:real;              {lenght of survey file (bytes)}
  spare:real;               {spare real variable on header}
  dis_form:char;            {display format W = wiggle, S = shade}
  points:integer;           {# of A-scans in T-saft data set}
  heaptop:pas;              {dynamic variable marker}
  blankbottom:integer;      {sample # of begining of last A scan blanking}

(*********** end of variable declarations ***************)


{$R+}
{$V-}

{These include files are defined in the Borland Turbo Graphix Toolbox
 manual and must be included in this order.  They are used in this program
 by procedure calls coded in the program.  For further information consult
 the Borland Turbo Graphix Toolbox manual.}
  {$I c: graphix typedef.sys}                      {variable declaration}
  {$I c: graphix graphix.sys}                      {definitions and routines}
  {$I c: graphix kernel.sys}                       {support routine}
  {$I c: graphix windows.sys}                      {graphics window manipulation

(******************************** DELAY ***********************************)
(*                                                                        *)
(*         assembly program for convolution operators (hyperbolas)        *)
(*                                                                        *)
(**************************************************************************)
PROCEDURE delay(c:integer;px0,vt,space,deep:real;VAR r:arr_r);
EXTERNAL 'delay.bin';


(******************************** FOLD2 ***********************************)
(*                                                                        *)
(*        assembly program for convolution image reconstruction           *)
(*                                                                        *)
(**************************************************************************)
PROCEDURE fold2(ls:integer;VAR sensor:arr_s;lr:integer;
                VAR r:arr_r;VAR lp:integer;VAR p:arr_p);
EXTERNAL 'fold2.bin';


(******************************** BEEP ************************************)
(*                                                                        *)
(*                           makes a sound                                *)
(*                                                                        *)
(**************************************************************************)
procedure beep(freq:integer);
begin
  sound(freq);
  for i := 1 to 10000 do i:=i;
  nosound;
end;    {beep}
```

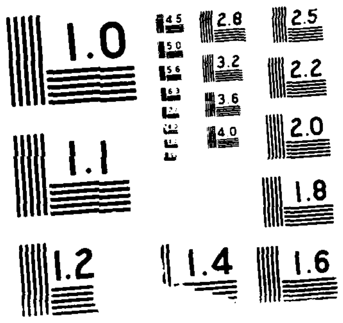
```
(****************************** TIMEDATE *********************************)
(*                                                                        *)
(*                           time and date                                *)
(*                                                                        *)
(**************************************************************************)
procedure timedate(var hour,min,month,day,year:integer);
type
  registers = record
              ax,bx,cx,dx,bp,si,di,ds,es,flags:integer;
            end;
var
  regs: registers;

begin
  with regs do            {time of day}
    begin
      ax := $2C00;
      msdos(regs);        {MS DOS function call to DOS time function}
      hour := hi(cx);
      min := lo(cx);
    end;
  with regs do            {date}
    begin
      ax := $2A00;
      msdos(regs);        {MS DOS function call to DOS date function}
      year := cx;
      month := hi(dx);
      day := lo(dx);
    end;
end;    {timedate}


(*************************** SET HEADER ***************************************)
(*                                                                            *)
(*     set global header parameters to the local input parameters values      *)
(*                                                                            *)
(******************************************************************************)

PROCEDURE SET_HEADER;
begin
     with info do                  {set header values}
          begin
                 nx     := trunc(points);
                 nz     := trunc(aslenv);
                 w      := width;
                 dt     := tsamp;
                 v      := vel;
                 sp     := calspace;
                 Xavg   := trunc(navg);
                 stn    := trunc(station);
                 ln     := lenght;
                 scns   := trunc(scans);
                 mrk    := trunc(delimiter);
                 zoffset := trunc(depthoff);
              dcap   := datcap;
              srcfile :=srcfile;
        END;
     sparename1:=fname;
end; {set header variables}

(***************************** SET VAR ****************************************)
(*                                                                             *)
(*  set program local variables to the global header parameter values          *)
(*                                                                             *)
(******************************************************************************)

PROCEDURE SET_VAR;
    BEGIN
        with info do                    {set header values}
          begin
              points := nx;
              aslenv := nz;
              width := w;
              tsamp := dt;
              vel := v;
              calspace := sp;
              navg := Xavg;
              station := stn;
              lenght := ln;
              scans := scns;
              delimiter := mrk;
              depthoff := zoffset;
              datcap := dcap;
              srcfile := srcfile;
      end;
      step      := char7;
      print     := char8;
      dis_form  := char9;
      scale     := spareI1;
      space     := calspace;
      vel0      := vel;
      if (step='Y')or(step='y')
        then stepping := true
        else stepping := false;
      if (print='Y')or(print='y')
        then printscreen := true                   {print output image}
        else printscreen := false;
      if scale > 15 then scale := 20
                 else scale := 10;

 END; {SET_VAR}


(********************************* ANNOTATE ***************************)
(*                                                                    *)
(*              write pertinent parameters to the screen              *)
(*                                                                    *)
(*********************************************************************)
```

```
procedure annotate;
var
  hour,
  min,
  month,
  day,
  year:integer;
begin
  timedate(hour,min,month,day,year);
  if (dis_form ='c') or (dis_form ='C')
   then
     begin
       gotoxy(33,2); write(fname);
       gotoxy(33,3); write(month:2,'-',day:2,'-',year mod 100:2);
       gotoxy(33,4); write(' ',hour:2,':',min:2);
       gotoxy(33,5); write('w=',space*(xsize-1):5:2);
       gotoxy(33,6); write('d=',aslenv*vt/2:5:1);
       gotoxy(33,7); write('o=',depthoff:5);
       gotoxy(33,8); write('v=',vel*1.0e-9:5:3);
       gotoxy(33,9); write('t=',tsamp*1.0e9:5:3);
       gotoxy(33,10); write('st=',station:4);
     end
   else
     begin
       timedate(hour,min,month,day,year);
       gotoxy(60,1); write(scrname);
       gotoxy(60,2); write(fname);
       gotoxy(60,3); write(hour:2,':',min:2,'  ',month:2,'-',day:2,'-',
                                                   year mod 100:2);
       gotoxy(60,4); write('width = ',space*(xsize-1):8:2);
       gotoxy(60,5); write('depth = ',aslenv*vt/2:4:1);
       gotoxy(60,6); write('dpthoffset = ',depthoff);
       gotoxy(60,7); write('vel(ft/nS) = ',vel*1.0e-9:5:3);
       gotoxy(60,8); write('tsamp(nS) =  ',tsamp*1.0e9:5:3);
     end;
end; {annotate}


(************************************ SETUP *******************************)
(*                                                                        *)
(*      scales and label axes of reconstructed image display              *)
(*                                                                        *)
(**************************************************************************)
Procedure setup;
var
  st: string[3];
  stnoff: integer;

BEGIN
  clrscr;
  {InitGraphic;}
  Entergraphic;
  DefineWindow(1,6,0,53,180);    {data window}
  Defineworld(1,0,0,383,180);    {32I * 12 = 384}
```

AD-A194 286 SIGNAL PROCESSING SOFTWARE FOR GROUND PENETRATING RADAR USER'S MANAAL(U) SIGMA RESEARCH INC REDMOND WA R LIEM ET AL MAR 88 NCEL-CR-88-010 N62474-85-C-4953 3/3

UNCLASSIFIED F/G 17/9 NL




1.0
4.5
5.0
5.6
6.3
7.1
8.0
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.2
1.4
1.6

```
  DefineWindow(2,0,0,5,190);      {verticle axis window}
  Defineworld(2,0,0,48,190);
  DefineWindow(3,0,181,79,199);  {horizontal axis window}
  Defineworld(3,0,0,639,19);
  Selectworld(2);
  SelectWindow(2);
  for i := 0 to scale do          {label vertical axis}
    begin
      drawline(44,i*vertpixels/scale+10,48,i*vertpixels/scale+10);
      str(i:2,st);
      if (not odd(i)) and (i > 0) then
        Drawtext(25,round(i*vertpixels/scale),1,st);
    end;    {for}
  Drawtext(25,2,1,' 0');
  Drawtext(0,64,1,'D');
  Drawtext(0,72,1,'e');
  Drawtext(0,80,1,'p');
  Drawtext(0,88,1,'t');
  Drawtext(0,96,1,'h');
  selectWorld(3);                 {label horizontal axis}
  selectwindow(3);
  stnoff := abs(station mod 100);
  for i:= 0 to 10 do
    begin
      drawline(i*384/10+48,15,i*384/10+48,19);
      if station >= 0
        then str(i+stnoff:3,st)
        else str(stnoff-i:3,st);
      DrawtextW(i*384/10+43,12,1,st);
    end;    {for}
  DrawtextW(150,5,1,'Station (ft)');
  DrawtextW(0,12,1,'sta ');
  str(abs(station div 100):2,st);
  DrawtextW(20,12,1,st);
  DrawtextW(32,12,1,'+');
  SelectWorld(1);                 {ready screen for data display}
  SelectWindow(1);
  SetBackground(0);
  Drawborder;
END;   {setup}


(***************************** FORMFEED ***********************************)
(*                                                                        *)
(*             makes printer go to next top of page                       *)
(*                                                                        *)
(**************************************************************************)
procedure formfeed;
var
  device: FILE of integer;
  task: integer;
  nz: integer;
```

```
BEGIN
  assign(device,'prn');           {setup control destination to printer}
  task := 12;                     {12 is the control for form feed}
  rewrite(device);
  write(device,task);
  close(device);
end;    {formfeed}


(**************************** WIGGLE PLOT *********************************)
(*                                                                        *)
(*             PLOTS FILLED-IN WIGGLE TRACE ON SCALED AXES                *)
(*                                                                        *)
(**************************************************************************)

PROCEDURE wiggleplot(image:arr_ix;imgx,imgd:integer);
VAR
  int,                       {plot intensity}
  ypix: integer;
  imin,                      {minimum image intensity}
  imax,                      {maximum image intensity}
  mag: real;                 {magnitude of image}
  i,j,m,n,k,l: integer;      {general indicies}
  x1,y1: integer;            {screen coordinate}
  st : string[1];
BEGIN
  setup;                                {initialize the screen}
  imax := 0.0;
  imin := 1.0E50;
  ypix := plty-1;
  FOR i:=0 TO imgd-1 DO                {determine maximum and minimum}
    BEGIN                              {image intensities in output image}
      FOR j:=0 TO imgx-1 DO
        BEGIN
          IF abs(image[j] [i])>imax
            then imax := abs(image[j] [i]);    {update maximum}
          IF abs(image[j] [i])<imin
            then imin := abs(image[j] [i]);    {update minimum}
        END;{loop j}
    END;    {loop i}

  k := 0;
  l := 0;
  int := 0;
  FOR j:=1 TO  vertpixels  DO
    BEGIN
      n := trunc((1.0*j)*((imgd )-1)/(ypix-1)*20/(vertpixels*vel*tsamp));
      FOR i:=1 TO xsize-1 DO       {n = column index of display image array}
        BEGIN
          m := i;                  {m = row index of display image array}
          IF (k<>m) OR (l<>n)      {if m and n changed then plot a point}
            THEN      {next point}
              BEGIN
```

```
                if n < aslen        {if addressing exhaust image array}
                  then              {then set output plot point to 0}
                    mag :=(image[m] [n]-imin)/abs(imax-imin)*(maxcolor)
                  else
                    mag := 0;
                if mag > maxcolor then  mag := maxcolor;
                int := round(mag);
                   begin   {draw image}
                      x1 := round((i*grlevels)*(space*(xsize-1))/10);
                                    {scales X position of output pixel}
                      y1 := vertpixels - j ;
                                    {calculates Y position of output pixel}
                      drawpoint(x1,y1);        {draw vert reference lines}
                      drawline(x1,y1,x1+int,y1);  {draw image pixel}
                   end;    {draw image}
              END;    {next point}
          gotoxy(60,20); write('x = ',i);     {write output image index}
          gotoxy(60,21); write('y = ',j);
          k := m;                        {update to next output pixel location}
          l := n;
        END;    {loop i}
    END;    {loop j}
  gotoxy(60,14);                         {annotates screen with max and min}
  writeln('imax=',imax:7);
  gotoxy(60,15);
  writeln('imin=',imin:8);
  gotoxy(60,18);
  writeln(' enter CR to cont  ');
  beep(110);
  scrname := fname;
  delete(scrname,pos('.',scrname),10);   {delete filename extension};
   if not stepping
    then scrname := scrname + '.img'      {prepare output image name}
    else begin
          str(iterate,st);
          scrname := scrname+'.im'+st;
        end;
  annotate;                         {add annotation of pertinent info onto screen}
    savescreen(scrname);              {save output image}
  if printscreen                      {print display}
    then
      begin
        inline   ($55/$cd/$05/$5d);        {prints screen}
        formfeed;                   {skip to next top of page}
      end
    else if not (stepping or (char6 ='Y') or (char6 = 'y')){char6 = multiproc}
                                    {if not auto velocity setting then}
          then readln;              {or multiprocessing wait until a CR is pressed}
  LeaveGraphic;                     {go back to text mode}
END;    {wiggleplot}


(****************************** COLOR PLOT *********************************)
```

```
(*                                                                         *)
(*               PLOT SHADED MAGNITUDE IN FOUR COLOR                       *)
(*                                                                         *)
(***************************************************************************)
PROCEDURE colorplot(image:arr_ix;imgx,imgd:integer);

  VAR
    int,xpix,ypix: integer;
    thresh,imax,imin,max,mag,it: real;
    rt,lt,tp,bt,i,j,m,n,k,l,ytick,ty: integer;

   BEGIN
     imax := 0.0;
     imin := 1.0E50;
     max := -1.0;
     xpix := pltx-1;
     ypix := plty-1;
     ytick := trunc((ypix+1)/ntick);
     lt := 0;
     rt := xpix;
     tp := 0;
     bt := ypix;
     FOR i:=0 TO imgd-1 DO
      BEGIN
        write('.');
        FOR j:=0 TO imgx-1 DO                   {seek max and min image values}
         BEGIN
           IF abs(image[j] [i])>imax
            THEN
            imax := abs(image[j] [i]);
           IF abs(image[j] [i])<imin
            THEN
            imin := abs(image[j] [i]);
         END;
      END;
     graphcolormode;                            {set color mode}
     graphbackground(0);
     palette(3);                                {select color combination}
     draw(lt,bt,lt,tp,maxshadecolor);           {draw border}
     draw(lt,tp,rt,tp,maxshadecolor);
     draw(rt,tp,rt,bt,maxshadecolor);
     draw(rt,bt,lt,bt,maxshadecolor);
     ty := ytick;
     k := 0;
     l := 0;
     int := 0;
     FOR j:=1 TO ypix-1 DO                             {color from top to bottom}
      BEGIN
        n := trunc((1.0*j)*(imgd-1)/(ypix-1));   {calculate image column index}
        FOR i:=1 TO xpix-1 DO
         BEGIN
           m := trunc((1.0*i)*(imgx-1)/(xpix-1));{calculate image row index}
           IF (k<>m) OR (l<>n)                   {if changed from previous}
            THEN                                 {then plot a point}
```

```
              BEGIN                                {scales intensity}
                mag := abs(abs(image[m] [n])-imin)*(2*maxshadecolor)
                                /abs(imax-imin)-1*maxshadecolor;
                IF mag>maxshadecolor               {limit max intensity}
                 THEN
                 mag := maxshadecolor;
                int := round(mag);
              END;
            IF int>=1
             THEN
             plot(lt+i,tp+j,int);                  {plot a pixel}
            k := m;
            l := n;
          END;
       END;
       gotoxy(33,18); writeln('imax=');            {annotate}
       gotoxy(33,19); writeln(imax:8);
       gotoxy(33,20); writeln('imin=');
       gotoxy(33,21); writeln(imin:8);
       gotoxy(33,23); writeln('enter CR');
      beep(110);
      annotate;
      if printscreen                   {print display}
       then
       begin
         inline   ($55/$cd/$05/$5d);        {prints screen}
         formfeed;                  {skip to next top of page}
       end
       else readln;                  {wait until a CR is pressed}
      textmode;
    END;          {colorplot}


(************************************ ASCN ********************************)
(*                                                                       *)
(*                    data and parameters input                          *)
(*                                                                       *)
(*************************************************************************)
PROCEDURE ascn(VAR sensor:arr_s;VAR tsamp,space:real;
               VAR points,zpoints:integer);
var a:integer;

BEGIN
  zpoints:=aslen;
  smax := 0;
  assign(data,fname);                   {open file fname}
  reset(data);
  blockread(data,ibuf,1);               {move pointer pass header}
  gotoxy(1,25);
  write('                                                                  ');
  gotoxy(1,25);
  write(' Reading ',fname,' Scan Data  ');
  blankbottom := aslen;
  {write('enter bottom data to supress [',aslen,' samples]  ');
```

```
readln(blankbottom);}             {experimental blanking}
for j := 0 to aslen-1 do avg[j] := 0; {clears average buffer}
FOR i:=0 TO 2*(points-1) DO         {read T-saft data and store into sensor}
  BEGIN                             {read one scan into data buffer}
    blockread(data,ibuf,(2*alen2+127) DIV 128);  {read data into buffer}
    new(sensor[i]);                 {reserve memory for one sensor scan}
    FOR j:=0 TO aslen-1 DO          {shift data according to depth offset}
      BEGIN
        if ((j + depthoff) < blankbottom) and ((j + depthoff) > 0 )
          then sensor[i] [j] := ibuf[j+depthoff]
          else sensor[i] [j] := 0;    {zero out of range data}
        IF j=0
          THEN sensor[i] [j] := 0;
        IF abs(sensor[i] [j])>smax             {keep tract of max amplitude}
          THEN smax := abs(sensor[i] [j]);
        avg[j] := avg[j] + sensor[i] [j];      {cummulate sensor[j] in avg}
      END;          {loop J}
  END;          {loop I}
write('I = ',I,' J = ',j);
write('  Max sensor int=',smax:4);             {write max data value}
close(data);

for j := 0 to aslen-1 do
  avg[j] := avg[j]/(2*points -1);     {Calculates avg sensor trace}
for i := 0 to 2*(points-1) do         {backgroound removal with limiting.}
  begin                               {The average trace as calculated above}
    for j := 0 to aslen-1 do          {is subtracted from each trace in the}
      begin                           {data set.  If at any point the data}
        x := sensor[i] [j];           {magnitude is less then the result is}
        a := round(avg[j]);           {limited to zero to prevent overshoot.}
        if ( x * a > 0 )              {overshoot limiting done here}
          then begin
                 if ( x > 0 )
                   then begin
                          if x - a > 0 then sensor[i] [j] := x-a
                                       else sensor[i] [j] := 0;
                        end;
                 if ( x < 0 )         {undershoot limiting done here}
                   then begin
                          if x - a < 0 then sensor[i] [j] := x-a
                                       else sensor[i] [j] := 0;
                        end;
               end;
      end;
  end;

(*           {blocked out code is for DC average removal
              without stationary background removal}
 for j := 0 to aslen-1 do avg[j] := avg[j]/(2*points -1); {avg of sensor[j]}
   for i := 0 to 2*(points-1) do
     begin
       for j := 0 to aslen-1 do        {average subtracted from data trace}
         sensor[i] [j] := sensor[i] [j] -round(avg[j]);
     end;
```

```
*)
END;      {ascn}


(****************************** SAFT ************************************)
(*                                                                      *)
(*       performs time domain synthetic aperture focusing               *)
(*       generates convolution operator and convolves withraw data      *)
(*                                                                      *)
(************************************************************************)
PROCEDURE saft(VAR vt,space:real;VAR sensor:arr_s;
               VAR asize,dsize:integer);
VAR
  p: arr_p;          {imtermediate output image array}
  r: arr_r;          {hyperbola delay table}
  halfd,             {= aslen/2 if scale=10 else = 20}
  l,k,i,j,lp,c: integer;
  halfa,            {half of asize; = points/2}
  dd,               {depth increment}
  px0,              {hyperbola apex location}
  deep:real;        {depth of current reconstruction iteration}

BEGIN     {saft}
  halfd:=dsize DIV trunc(20/scale);     {if halfd = dsize image is 400 points}
                                        {20 is the 20 ft full vert scale}
  halfa := asize/2.0;
  dd := vt/2.0;                         {depth increment}
  px0 := space*(halfa-1);               {apex of operator hyperbola}
  deep := 0.0;                          {initial operator depth}
  FOR i:=0 TO halfd-1 DO                (****** 400 image pts vert *****)
    BEGIN
      FOR k:=0 TO asize-1 DO r[k]:=0;   {clears arr r}
      write(' ',i);
      delay(asize,px0,vt,space,deep,r); {generate hyperbola operator}
      FOR k:=0 TO asize-1 DO
        BEGIN                         {zero operator wings beyond avialable data}
          IF r[k]>(aslenv-1)
            THEN r[k] := 0;
        END;
      fold2(asize,sensor,asize-1,r,lp,p);    {convolution matrix operations}
      FOR j:=0 TO asize-1 DO
        begin
          image[j] [i] := p[j+asize-1];      {select middle portion of recon.}
        end;                                 {for image dilplay}
      deep := dd+deep;                       {increment reconstruction depth}
    END;  { for loop I}
  BEGIN
    beep(220);                               {alert completion of recon.}
    if (dis_form ='c') or (dis_form = 'C')
      then colorplot(image,asize,halfd)      {four color display}
      else wiggleplot(image,asize,halfd);    {wiggle display}
    writeln;
  END;
```

```
END;    {saft}

(******************** PARAMETER INPUT ****************************************)
(*                                                                           *)
(*   procedure to input file name and data parameter, used only in manual    *)
(*   input mode ( no menu).                                                  *)
(*                                                                           *)
(*****************************************************************************)

PROCEDURE PARMIN;

BEGIN
  write('Enter A-scan data file name----');
  readln(fname);
  if fname = '' then fname := 'demo.sft';       {default filename }
  assign(data,fname);
  reset(data);
  blockread(data,buff,1);                  {read header information}
  move(buff,info,sizeof(info));
  close(data);
  write('enter scale 10 or [20] feet  ');  {vertical display scale}
  readln(scale);
  if scale >15 then scale := 20
               else scale := 10;
  with info do                        {set header values}
    begin
      points := nx;
      aslenv := nz;
      width := w;
      tsamp := dt;
      vel := v;
      calspace := sp;
      navg := Xavg;
      station := stn;
      lenght := ln;
      scans := scns;
      delimiter := mrk;
    {  aslenv := alen;   }
      datcap := dcap;
    {  fin := srcfile;   }
    {  spare := vl ;     }
      depthoff := zoffset;
      {for i := 1 to 2 do
         datafilenames[i] := ''; }
      space := w/(points-1);
    END;
                                  {allow override station entry}
  write('station is  ',station,' ft; enter change  ');
  readln(station);
  writeln('Time sampling interval  =',tsamp*1e9:10:3,' (nanoseconds)');
  writeln('Number of array points  =',points:10);
  writeln('Array element spacing   =',space:10:3,' (ft)');

  {depthoff := zoffset;}                        {default depth offset}
  write('Depth offset ',depthoff, ' samples; enter change  ');
  readln(depthoff);
  xsize := points;
  zsize := aslen;
  vel0 := vel*1.0e-9;                            {initial velocity}
  write('velocity = ',vel0:6:4, 'ft/nS; enter change  ');
  readln(vel0);
  vel0 := vel0 * 1.0e9;
  write(' Velocity stepping ? Y or [N] ');
  readln(step);
  if (step='Y')or(step='y')
    then stepping := true
    else stepping := false;
  write('display format: unscaled 4 color (C) or scaled wiggle [W]? ');
  readln(dis_form);
  write(' Print output image? Y or [N]');
  readln(print);
  if (print='Y')or(print='y')
    then printscreen := true                  {print output image}
    else printscreen := false;

 END;
(******************** LIST VARIABLES ***********************************)
(*                                                                     *)
(***********************************************************************)

procedure list_Var;
  Begin
      writeln;
      writeln('  points   := ',points:8);
      writeln('  aslen    := ',aslen:8);
      writeln('  width    := ',width:8:3);
      writeln('  tsamp    := ',tsamp*1e9:8:4);
      writeln('  vel      := ',vel*1e-9:8:4);
      writeln('  calspace := ',calspace:8:4);
      writeln('  navg     := ',navg:8);
      writeln('  station  := ',station:8);
      writeln('  lenght   := ',lenght:8:0);
      writeln('  scans    := ', scans:8);
      writeln('  delimiter:= ',delimiter:8);
      writeln('  aslen    := ',aslen:8);
      writeln('  datcap   := ',datcap:8:0);
      writeln('  depthoff := ',depthoff);
      writeln;
   End;

(**************************** INITIALIZE *********************************)
(*                                                                       *)
(*************************************************************************)


procedure initialize;
begin
  grlevels := 12;              {output magnitude display levels}
```

```
    scale:=20;                    {default vertical scale = 20 ft }
    station := 0;                 {default station}
    step:='N';                    {default is no velocity setpping}
    print:='n';                   {no printing of reconstruced image}
    dis_form:='w';                {default is wiggle format display}
  end;


  (******************************** MAIN ***********************************)
  (*                                                                       *)
  (*************************************************************************)

  BEGIN
  Mark(heaptop);              {set heaptop to current heap poniter value}
  initGraphic;
  LeaveGraphic;
    initialize;                 {initialize default values}
    which_screen := 2;          {2 = screen for tsaft}
    chain_return::=true;        {set return from workhorse flag}
    if auto_processing = 'Autoprocess'
    then begin
           spareI1 := 20;      {vertical display scale to be used in autoprocess}
           sparename1 := fname; {save fname transferred from Transfwk}
         end;
    set_var;                   {proc sets local var to globals passed from menu}
  {  parmin;                   {parameter input for individual prog execution}
  {  list_var;                 {list the variables for checking}

    xsize := points;
    ascn(sensor,tsamp,space,xsize,zsize);      {input data and parameters}
    FOR i:=0 TO xsize-1 DO                     {reserve memory for output image}
      begin
        new(image[i]);
      end;
    clrscr;
    IF vel0>0.0 then                          {start saft processing}
      BEGIN
        if stepping                        {multi stepped velocity processing}
        then
          begin   {stepping}
            for Iterate := 0 to 4 do
              begin
                vel := vel0*(0.90 +0.05*Iterate);    {increment velocity by 5%}
                writeln('processing ',fname,'  vel0=',vel0:8,'   vel=',vel:8);
                vt := vel*tsamp;
                saft(vt,space,sensor,xsize,zsize);   {invoke saft routine}
              end;{do iterate}
              vel := vel0;
           end    {stepping}
        else                                       {single pass processing}
          begin
            vel := vel0;
            writeln('processing ',fname,'  vel0=',vel0:8,'   vel=',vel:8);
            vt := vel*tsamp;
            saft(vt,space,sensor,xsize,zsize);  {invoke saft routine}
          end;
      END;    {IF}
      leavegraphic;
      release(heaptop);  {discard dynamic variables and reset heap pointer}
      clrscr;
      set_header;
  (*    halt;  *)        {test exit}
      assign(workhorse,'tsaft.chn');
      chain(workhorse);

  END.     {WORK HORSE PROGRAM T-SAFT}
```

```
;****************************************************************:
; This procedure is used to generate the convolution operator    :
; in the T-SAFT delay and sum. The inputs are the physical       :
; parameters of the hyperbola and the output of delay values     :
; that define a hyperbola at the reconstruction depth.           :
; This program is called once for each depth depth reconstructed. :
;                                                                :
; This procedure is invoked by the following statement:          :
;                                                                :
;      PROCEDURE delay(c:integer;px0,vt,space,d:real;            :
;                      VAR r:array[0..c-1] of integer);          :
;                                                                :
;****************************************************************:



        .8087                   ;invoke math coprocessor
locvar  equ     2
locstk  struc
      - dw      (8000H-locvar/2) dup (1234h)
i       dw      101h
locstk  ends
;
delseg  struc
        dd      ?
r       dd      ?               ;pointer to array r
d       dq      ?               ;depth of hyperbola
space   dq      ?               ;trace to trace spacing
vt      dq      ?               ;vertical sample depth
px0     dq      ?               ;half hyperbola width
c       dw      ?               ;hz # of points in hyperbola (points)
delseg  ends
;
code    segment byte public 'CODE' ;declare code as public
        assume cs:code
        public  delay
;
delay   proc    near
;       enter   locvar,0
        mov     i[bp],0                 ;clears [bp]
        mov     cx,c[bp]                ;asize DIV 2   (asize=points)
        shr     cx,1                    ;initial loop ctr value = asize/2
        mov     ax,c[bp]                ;asize-2
        sub     ax,2
        mov     c[bp],ax
        les     bx,r[bp]
del0:   push    cx                      ;put loop ctr on stack
        fild    [bp].i                  ;put i on stack top
        fmul    [bp].space              ;ST = i*space
        fsub    [bp].px0                ;ST = i*space - px0
        fmul    st,st(0)                ;ST*ST = X**2
        fld     [bp].d                  ;ST = depth
        fmul    st,st(0)                ;ST = d**2
        fadd                            ;ST = X**2+d**2
        fsqrt                           ;ST = sqrt(X**2+d**2)
        fdiv   [bp].vt                  ;ST = sqrt(X**2+d**2)/vt
        fadd   st,st(0)                 ;ST = 2ST  round trip
        mov     si,i[bp]
        shl     si,1
        fistp es:word ptr[bx+si]        ;store 2sqrt(x**2+d**2)/vt
        fwait                           ;    into array r
        mov     ax,es:word ptr [bx+si]
        mov     si,c[bp]
        sub     si,i[bp]
        shl     si,1
        mov     es:word ptr [bx+si],ax ;
        inc     i[bp]                   ;increment index i
        pop     cx                      ;recall loop ctr
        loop    del0                    ;decrement ctr, exit if 0
;        leave                          ;else goto del0
        ret     (size delseg)-4         ;return
delay   endp
code    ends
        end
```

```
;************************ FOLD.ASM CONVOLUTION **************************
;  This program is used to convolve the operator hyperbola generated    :
;  by DELAY.ASM with the radar data. Effectively it takes each point    :
;  of intersection of the operator and the data and sum them to make    :
;  one reconstructed image value at the apex of the hyperbola. The      :
;  operator is moved one sample over and the process repeated to make   :
;  this next.  When the whole row is completed DELAY.ASM generates      :
; another operator hyperbola one sample down and the process is         :
; repeated.                                                             :
;                                                                       :
;  The calling statement is as follows:                                 :
;                                                                       :
;   PROCEDURE fold(la:integer;a:arrS;lb:integer;                        :
;                       b:arrR;lc:integer;VAR c:arrP);                  :
;                                                                       :
;************************************************************************

;  (****) denotes changes made to run 2 wide data 10-21-86.

.8087                           ;invoke math coprocessor
;public  c,lc,b,lb,a,la
public  fold0,fold1,fold2,fold3,fold4
locvar  equ     2
locstk  struc
        dw      (4000H-locvar/2) dup (?)
i       dw      ?
locstk  ends
folseg  struc
        dd      ?               ; spacer only
c       dd      ?               ; pointer to array c, c = output array
lc      dd      ?               ; pointer to lc     , not used here
b       dd      ?               ; pointer to array b, b = delay table
lb      dw      ?               ;              , lb = asize-1 = nx-1 = points-1
a       dd      ?               ; pointer to array a, a = sensor = input array
la      dw      ?               ;              , la = asize = nx = points
folseg  ends
;
code    segment byte public 'CODE'
        assume cs:code
        public  fold
;
fold    proc    near
;        enter   locvar,0
        push    bp
        mov     bp,sp
        sub     sp,locvar
        mov     ax,la[bp]     ; setup to clear output array c
        add     ax,lb[bp]     ; array size = asize + 2*(asize-1)
        add     ax,lb[bp]     ; (*******)
        mov     cx,ax         ; cx = array size
        les     di,c[bp]      ; load addr of arr c
        shl     cx,1          ; multiply array size by 4
        shl     cx,1          ;
        mov     ax,0          ; load 0 into ax for clearing c
        cld                     ; clears direction flag
        rep     stosw           ; clears entire arr c by storing 0's
        mov     i[bp],0         ; clears i, i is the major index
        mov     ax,la[bp]       ; setup outer loop iterations   (**)
        add     ax,lb[bp]       ; count = no of A-scans in      (**)
        mov     cx,ax           ;         data set              (**)
fold0:  push    cx              ; save start count
        mov     di,i[bp]
        mov     cl,3
        shl     di,cl
        mov     cx,lb[bp]       ; set hyp operator loop ctr = asize-1
        mov     si,0
fold1:  push    cx              ; save hyp opr ctr
        push    si
        les     bx,b[bp]        ; load delay table addr
        mov     ax,es:word ptr [bx+si]
        mov     si,i[bp]
;        shl     si,2
        shl     si,1
        shl     si,1
        les     bx,a[bp]        ; load sensor ptr addr
        les     bx,es:dword ptr [bx+si]   ; load sensor data addr
        shl     ax,1
        mov     si,ax
        fild    es:word ptr [bx+si]   ; load data point with delay into stack
        fabs                            ; abs value
        fsqrt                           ; take square root
        test    es:word ptr [bx+si],8000H   ; check original polarity
        jz      fold2                   ; skip next if pos
        fchs                            ; change sign
fold2:  pop     si
        les     bx,c[bp]
        fadd    es:qword ptr [bx+di]    ; add to cummulative
        fstp    es:qword ptr [bx+di]    ; store image value in memory
        add     di,8                    ; move output ptr to next mem loc
        add     si,2                    ; move input ptr to next input
        pop     cx
        loop    fold1                   ; loop until hyperbola is exhauste
        inc     i[bp]
        pop     cx
        loop    fold0                   ; loop until row is exhausted
        mov     cx,la[bp]               ; setup loop ctr for squaring
        add     cx,lb[bp]               ; output array elements
        add     cx,lb[bp]               ; ctr = 3asize-3     (**)
        sub     cx,1
        mov     si,0
        les     bx,c[bp]
fold3:                          ; square output but preserve polarity
        fld     es:qword ptr [bx+si]      ; load image value
        fmul    st,st(0)                  ; square it
        test    es:word ptr [bx+si+6],8000H   ; test polarity
        jz      fold4                     ; skip next stm if positive
        fchs                              ; change sign
fold4:
        fstp    es:qword ptr [bx+si]         ; store intensity back
        add     si,8                         ; set index for next image value
        loop    fold3                        ; loop to fold3 until done
;        leave
        mov     sp,bp
        pop     bp
        ret     (size folseg)-4
fold    endp
code    ends
        end
```

```
(******************** VELOCITY DETERMINATION PROGRAM *************************)
(*                                                                          *)
(*   This velocity determination program is designed to help extract the    *)
(*   radar propagation velocity and the depth offset in the radar data.     *)
(*   These parameters can then be input into T-saft for the image recon-    *)
(*   struction.  The input requirements of this program are the digitization *)
(*   sampling period and the survey width represented by the data.          *)
(*       The program is self guided with prompts for inputs.  First, the    *)
(*   width is calibrated, then the program generates a hyperbola using these *)
(*   calibration data.  The hyperbola can be overlaid onto the hyperbola    *)
(*   composed by the multiple traces of the radar data.  The shape and      *)
(*   location of the template hyperbola is manipulated with the keys of     *)
(*   the numeric keypad.  The template parameters are displayed at the      *)
(*   bottom of the screen.The arrow keys translates the template.  The      *)
(*   Home and End keys increases and decreases the velocity of the template *)
(*   hyperbola.  The Page up and the Page down keys increases and decreases *)
(*   the depth of the template hyperbola.                                   *)
(*       The goal is to make a perfect match between the template and the   *)
(*   data hyperbolas; when this is accomplished the velocity and the depth  *)
(*   offset parameters of the template are taken as the data specifications. *)
(*   This program is designed to be flexible and can work with both the     *)
(*   Preview and the Tsaft data display and can be calibrated in both the   *)
(*   vertical and the horizontal dimensions.                                *)
(*                                                                          *)
(*                                                     7-1-87               *)
(****************************************************************************)

program Velocity_Determination;


{ Global Required Definition of variables and constants }

    TYPE                                {writeheader variables}
          id = RECORD
                 nx,                    {nx= # of A-scans in one data set}
                 nz     :integer;       {nx = A-scan lenght, normally 400}
                 w,                     {w = survey width}
                 dt,                    {dt = sampling period}
                 v      :real;          {v = radar propagation velecity}
                 sp     :real;          {sp = averaged data scan-scan spacing}
                 Xavg   :integer;       {Xavg = # of A-scan averaged per data A-scan}
                 stn    :integer;       {station = data station}
                 ln     :real;          {ln = data lenght}
                 scns   :integer;       {scns =# of A-scansin the data}
                 mrk    :integer;       {delimiter between A-scans in original data}
                 zoffset:integer;       {template determined sample offset}
                 dcap   :real;          { data capacity of a disk }
                 srcfile:string[40];{scource data file}
                 spare1:real;           {template determined velocity}
                 spare2:integer;        {spare}
               END;
    VAR    { global variables }
          fname: string[40];         {name of I/O file}
          fin: string[40];           {name of input file}
```

```
        workhorse:file;          {chainfile assignee}
        fout:string[40];         {name of output file}
        which_screen:integer;    {which screen to use }
        chain_return:boolean;    {returning from chain program?}
        auto_processing:string[14];{automatic disk processing key}
        {the following characters are used for program flow control}
        char1,                   {plot intensity var in Preview}
        char2,                   {width calibtation var in Preview}
        char3,                   {print var in Preview}
        char4,                   {print var in Transfer}
        char5,                   {disk process var in Transfer}
        char6,                   {multiple processing var in T-saft}
        char7,                   {velocity stepping var in T-saft}
        char8,                   {print image var in T-saft}
        char9,                   {display format var in T-saft}
        char10,              {spare character variables}
        char11,              {spare character variables}
        char12,              {spare character variables}
        char13,              {spare character variables}
        char14,              {spare character variables}
        char15:char;         {spare character variables}
        sparename1,
        sparename2,
        sparename3,
        sparename4:string[40];   {spare strings}
        spareI1,
        spareI2,
        spareI3,
        spare14:integer;         {spare integers}
        spareR1,
        spareR2,
        spareR3,
        spareR4:real;            {spare reals}
        info:id;                 {header information}

{the above are global variables to be passed from progran to program during
 the chain calls. These variables are to be declared exactly as above in all
 programs wether used or not.  Tailoring will be done with the spares.}


TYPE
     Num = byte;
     heappionter =  integer;       {heap pointer}

VAR
     {fname:string[25];}           {name of I/O file}
     {fin: string[25];}            {name of input file}
     {fout:string[40];}            {output data file name}
     buff: array[0..127] of num;   {input parameter buffer}
     fdata: FILE;                  {output disk file}
     datfile: file of num;         {raw data disk file}
     heaptop : heappionter;        {heap pointer}
```

```
{These include files are defined in the Borland Turbo Graphix Toolbox
 manual and must be included in this order.  They are used in this program
 by procedure calls coded in the program.  For further information consult
 the Borland Turbo Graphix Toolbox manual.}
  {$I typedef.sys}                        {variable declaration}
  {$I graphix.sys}                        {definitions and routines}
  {$I kernel.sys}                         {support routine}
  {$I windows.sys}                        {graphics window manipulation routine}
  {$I polygon.hgh}                        {polygon generation routine}
  {$I modpoly.hgh}                        {polygon modification routine}

var
  ch: char;                             {program flow control input variable}
  Zexp:real;                            {depth expansion factor in screen data}
  n,                                    {# of vertex points in template}
  ind: integer;                         {vertices listing index}
  hyperbola: PlotArray;                 {the template hyperbola}
  z0,                                   {equivalent depth of template}
  CurrX,                                {x position of template}
  CurrY,                                {y poxition of template}
  IncrX,                                {x increment to translate}
  IncrY,                                {y increment to translate}
  Size,                                 {size of template hyperbola}
  speed: real;                          {parameter increment multiplier}
  vel: real;                            {radar propagation velocity}
  dx,                                   {horizontal unit distance}
  width,                                {width between marked positions}
  tsamp: real;                          {digitization sampling time}
  calib: boolean;                       {control variable for calibration}
  OK:boolean;                           {file existence flag}


(******************************** BACK ************************************)
(*                                                                        *)
(*                      generate the background                           *)
(*                                                                        *)
(**************************************************************************)
procedure back;
begin
  ClearScreen;
  SetBackground(0);                       {select black background}
 (* writeln('enter complete filename ');  {input screen file name}
  readln(fname);
  fname := fname;  *)                     {blocked for integration with menu}
  loadscreen(fname);                      {loads the background data image}
  DrawBorder;
  gotoxy(1,24);                           {clears bottom two lines}
  writeln('                                                                   ',
         '                   ');
  write('                                                                   ',
       '                   ');
  {wait until a key is pressed}
  CopyScreen;                             {copies image into secondary screen}
end;    {back}
```

```
(************************* MAKEHYPERBOLA ********************************)
(*                                                                      *)
(*                       generate a hyperbola                           *)
(*                                                                      *)
(************************************************************************)
procedure Makehyperbola(var hyperbola:PlotArray;n:integer;z0:real);
var
  i:integer;
  delta:real;

begin
  delta:=size/(n-1);               {hyperbola verticies step size}
  FOR i:=1 TO n DO                 {calculate the verticies coordinates}
    begin
      hyperbola[i,1]:=(i-1)*delta-size/2;
      hyperbola[i,2]:= ((2.0*sqrt(sqr(dx*hyperbola[i,1])+sqr(z0))
                                         /(tsamp*vel)))*Zexp;
    end;
end;    {Makehyperbola}


(************************* RESTOREBACK **********************************)
(*                                                                      *)
(*   redraws the data plot (background) then draw the hyperbola in the *)
(*   newly calculated position and shape                                *)
(*                                                                      *)
(************************************************************************)
procedure Restoreback;
begin
  selectscreen(2);                      {copies a fresh display from}
  copyscreen;                           {inactive screen}
  selectscreen(1);
  DrawPolygon(hyperbola,1,-n,4,2,0);    {draw hyperbola with connected}
  setcolorblack;                        {rectangular verticies}
  DrawPolygon(hyperbola,1,-n,-4,1,0);   {hollow out verticies}
  setcolorwhite;
end;    {Restoreback}


(**************************** MoveUp ************************************)
(*                                                                      *)
(*                         move polygon up                              *)
(*                                                                      *)
(************************************************************************)
begiprocedure MoveUp(speed:integer);
    IncrX := 0;
    IncrY := 1*speed;
    CurrX:=CurrX+IncrX;                 {calculate the hyperbola apex pos}
    CurrY:=CurrY-IncrY;
    TranslatePolygon(hyperbola,-n,IncrX,-IncrY); {calculate vertices for}
    Restoreback;                                 {new position}
  end;    {MoveUp}
```

```
(***************************** MoveDown ***********************************)
(*                                                                        *)
(*                          move polygon down                             *)
(*                                                                        *)
(**************************************************************************)
procedure MoveDown(speed:integer);
begin
  IncrX := 0;
  IncrY := 1*speed;
  CurrX:=CurrX-IncrX;                  {calculate the hyperbola apex pos}
  CurrY:=CurrY+IncrY;
  TranslatePolygon(hyperbola,-n,-IncrX,IncrY); {calculate vertices for}
  Restoreback;                                 {new position}
end;    {MoveDown}


(***************************** MoveLeft ***********************************)
(*                                                                        *)
(*                          move polygon left                             *)
(*                                                                        *)
(**************************************************************************)
procedure MoveLeft(speed:integer);
begin
  IncrX := -1*speed;
  IncrY := 0;
  CurrX:=CurrX-IncrX;                  {calculate the hyperbola apex pos}
  CurrY:=CurrY-IncrY;
  TranslatePolygon(hyperbola,-n,IncrX,IncrY);  {calculate vertices for}
  Restoreback;                                 {new position}
  if calib then write(currx);
end;    {MoveLeft}


(**************************** MoveRight ***********************************)
(*                                                                        *)
(*                          move polygon right                            *)
(*                                                                        *)
(**************************************************************************)
procedure MoveRight(speed:integer);
begin
  IncrX :=-1*speed;
  IncrY := 0;
  CurrX:=CurrX+IncrX;                  {calculate the hyperbola apex pos}
  CurrY:=CurrY+IncrY;
  TranslatePolygon(hyperbola,-n,-IncrX,-IncrY);  {calculate vertices for}
  Restoreback;                                 {new position}
  if calib then write(currx);
end;    {MoveRight}


(******************************* zOup *************************************)
(*                                                                        *)
(*                     decrease hyperbola depth                        *)
(*                                                                     *)
(***********************************************************************)
procedure z0up(var n:integer;speed:integer;var z0:real);
begin
  z0 := z0 -speed*tsamp*vel;                         {decrease depth}
  if z0 < 0 then z0 := 0;                            {limit to zero depth}
  makehyperbola(hyperbola,n,z0);                     {recalculate new verticies}
  TranslatePolygon(hyperbola,-n,-currX,CurrY); {calculate vertices for}
  Restoreback;                                       {new position}
end;    {z0up}


(******************************* z0dn ***********************************)
(*                                                                     *)
(*                     increase hyperbola depth                        *)
(*                                                                     *)
(***********************************************************************)
procedure z0dn(var n:integer;speed:integer;var z0:real);
begin
  z0 := z0 + speed*tsamp*vel;                        {increment depth}
  makehyperbola(hyperbola,n,z0);                     {recalculate new verticies}
  TranslatePolygon(hyperbola,-n,-currX,currY); {calculate vertices for}
  Restoreback;                                       {new position}
end;    {z0dn}

(******************************* Vup ************************************)
(*                                                                     *)
(*                     increase radar velocity                         *)
(*                                                                     *)
(***********************************************************************)
procedure Vup(var n:integer;speed:integer;var z0:real);
begin
  Vel := vel + speed*1e6;
  makehyperbola(hyperbola,n,z0);                     {recalculate new verticies}
  TranslatePolygon(hyperbola,-n,-currX,CurrY); {calculate vertices for}
  Restoreback;                                       {new position}
end;    {Vup}


(******************************* Vdn ************************************)
(*                                                                     *)
(*                     reduce radar velocity                           *)
(*                                                                     *)
(***********************************************************************)
procedure Vdn(var n:integer;speed:integer;Var z0:real);
begin
  Vel := vel - speed*1e6;
  makehyperbola(hyperbola,n,z0);                     {recalculate new verticies}
  TranslatePolygon(hyperbola,-n,-currX,CurrY); {calculate vertices for}
  Restoreback;                                       {new position}
```

```
end;    {Vdn}


(******************************* cal ***************************************)
(*                                                                         *)
(*      calibrate screen to actual survey width                            *)
(*                                                                         *)
(***************************************************************************)
procedure cal;
var
  cal_message:string[40];
  x1,x2:real;

begin
  calib:= true;
  gotoxy(3,24);
  cal_message := 'move peak to left mark,depress  F1       ';
  repeat
    gotoxy(3,24);
    write(cal_message);
    while keypressed do read(Kbd,Ch);     {read the keystroke}
    case ord(Ch) of
      72 : MoveUp(1);                     {up arrow?}
      75 : MoveLeft(1);                   {left arrow?}
      77 : MoveRight(1);                  {right arrow?}
      80 : MoveDown(1);                   {down arrow?}
      56 : MoveUp(15);                    {shift up arrow?}
      52 : MoveLeft(20);                  {shift left arrow?}
      54 : MoveRight(20);                 {shift right arrow?}
      50 : MoveDown(15);                  {shift down arrow?}
{F1}  59 : begin                          {note begining x coordinate}
             cal_message := 'move peak to right mark, depress  F2          ';
             x1 := currX; write(x1:5:0);
           end;
{F2}  60 : begin                          {note end x coordinate}
             cal_message := 'enter distance in feet between marks     ';
             gotoxy(3,24);
             write(cal_message,'                              ');
             x2 := currX;
             gotoxy(45,24);
             readln(width);               {width to scale display}
             if (x2 - x1) = 0  then x2 := x2 + 1; {avoid division by 0}
             dx := width/(x2-x1);         {scale display to survey width}
             gotoxy(50,24);
             write('  dx = ',dx:7:4);
             gotoxy(3,25);
             write('enter radar sampling period (nS) ');
             readln(tsamp);
             tsamp := tsamp * 1.0e-9;     {note:tsamp is always displayed and}
             gotoxy(45,25);      {entered in nS but stored in actual magnitude}
             write('Press  F3  to complete calibration');
           end;
{F3}  61 : calib := false;                {F3 exits calibration}
```

```
    end;
    if ord(ch) <> 61 then ch := 'a';      {set to arbitary value <> 'F3'}
  until ord(ch) = 61;
end;    {cal}


(********************************* BLINK *********************************)
(*                                                                        *)
(*            makes the hyperbola blink to increase visibility           *)
(*                                                                        *)
(**************************************************************************)
procedure blink;
begin
  delay(100);
  setcolorblack;
  DrawPolygon(hyperbola,1,-n,-4,2,0);      {draw a black hyperbola}
  setcolorwhite;
  DrawPolygon(hyperbola,1,-n,-4,1,0);      {draw a white hyperbola}
  delay(100);
  DrawPolygon(hyperbola,1,-n,-4,2,0);      {draw a white hyperbola}
  setcolorblack;
  DrawPolygon(hyperbola,1,-n,-4,1,0);      {draw a black hyperbola}
  setcolorwhite;
end;    {blink}


(****************************** INITIALIZE ******************************)
(*                                                                        *)
(**************************************************************************)
procedure initialize;
begin
  Size:=300;                   {size of template hyperbola}
  n := 15;                     {number of point to generate for the hyperbola}
  CurrX:=0;                    {x pos of hyperbola apex in world coordinate}
  CurrY:=0;                    {y pos of hyperbola apex at 0 depth in world coor}
  dx := 0.07;                  {arbitary initial pixel distance}
  z0 := 0;                     {depth in feet of hyperbola}
  tsamp:= 1.6e-10;             {sample time of digitized radar data}
  vel:= 4.00e8;                {radar propagation velocity}
  Zexp := 1.0;                 {depth scale factor of data screen}
  calib:=true;                 {default setting is to calibrate width}
  InitGraphic;                 {initialize the graphics system}
  setlinestyle(0);               {select unbroken lines}
  DefineWindow(1,0,0,79,180);    {display screen size}
  DefineWorld(1,0,0,639,399);    {coordinates limits}
  SelectWorld(1);                {select it's world}
  SelectWindow(1);               {select window}
  SetBackground(0);              {give it a black background}
end;  {initialize}


(*********************** PROCEDURE WRITEHEADER ****************************)
(*                                                                          *)
(*      write pertinent data parameters in data file header                 *)
(*                                                                            *)
(******************************************************************************)
Procedure writeheader;
    begin

      reset(fdata);

     with info do                                     {set header values}
          begin                                       {updates only these}
                 dt := tsamp;                         {three variables}
                 v := vel;
                 zoffset := round(currY);
           END;


      blockwrite(fdata,info,(sizeof(info)+127) DIV 128);  {write it}
    end;    {writeheader}


(**************************** Main program body ****************************)
(*                                                                         *)
(***************************************************************************)
begin
  mark(heaptop);             {save heap pointer for later memory release}
  Initialize;                                     {initialize variables}
  back;                           {generate the radar scan background}
  Makehyperbola(hyperbola,n,z0);
  gotoxy(3,24);
  write('enter depth expansion factor [1]');  {depth scale factor used to }
  readln(Zexp);           {accomodate screens written at different scales}
  gotoxy(3,24);
  writeln;
  setlinestyle(1);
  DrawPolygon(hyperbola,1,-n,4,3,0);            {draw initial hyperbola}
  setcolorblack;
  DrawPolygon(hyperbola,1,-n,-4,2,0);
  setcolorwhite;
  cal;                                            {do calibration}
  repeat
    while not keypressed do blink;
    while keypressed do read(Kbd,Ch);           {read the keystroke}
    case ord(Ch) of
      72 : MoveUp(1);                {up arrow ?}
      56 : MoveUp(15);               {shift up arrow ?}
      80 : MoveDown(1);              {down arrow ?}
      50 . MoveDown(15);             {shift down arrow ?}
      75 : MoveLeft(1);              {left arrow ?}
      52 : MoveLeft(25);             {shift left arrow ?}
      77 : MoveRight(1);             {right arrow ?}
      54 : MoveRight(25);            {shift right arrow ?}
      73 : z0up(n,1,z0);             {page up key ?}
      57 : z0up(n,4,z0);             {shift page up key ?}
      81 : z0dn(n,1,z0);             {page down key ?}
```

```
         51 : z0dn(n,4,z0);             {shift page down key ?}
         71 : Vup(n,5,z0);              {home key ?}
         55 : Vup(n,20,z0);             {shift home key ?}
         79 : vdn(n,5,z0);              {end key ?}
         49 : vdn(n,20,z0);             {shift end key ?}
         62 : for ind := 1 to n do      {F4 list hyperbola coordinates}
                begin
                  gotoxy(65,ind);
                  if n <= 25 then
                    write(hyperbola[ind,1]:5:2,' ',hyperbola[ind,2]:5:2);
                end;    {for}
   {F9}  67 : begin                     {update data file parameters}
                 gotoxy(1,25);
                 writeln('                                                   ');
                 write(' enter file to update  ');
                 readln(fout);
                   while pos(' ',fout)=1 do  {remove leading blanks}
                   delete(fname,1,1);
                   assign(fdata,fout);
                   {$I-} reset(fdata) {$I+};
                   OK := (IOresult = 0);
                   if not OK
                 then  begin              {cannot find header file}
                          gotoxy(1,25);
                          write('Cannot find header file ',fout,' wait        ');
                          delay(4000);
                       end
                 else                     {header file is good}
                   begin
                   blockread(fdata,buff,1);    {read header information}
                   move(buff,info,sizeof(info)); {copy header into info}
                   close(fdata);      {the following update the display entries}
                   writeheader;
                   close(fdata);
                   end;
               end;
      end;    {case}
      if ord(ch) <> 68 then ch := 'a';    {F10?}
      gotoxy(3,24);
      writeLn(' x = ',currX*dx:6:2,'  depthoffset =  ',currY:5:0,
              '  vel = ',vel*1e-9:6:4,'  z0 = ',z0:6:3);
      gotoxy(1,25);
      writeln(' Press  F9  to update file parameters,  F10  to quit   ');
    until ord(Ch)=68;                   {F10 exits velocity program}
    LeaveGraphic;                       [leave the graphics system}
    release(heaptop);                   {give up heap space}
      clrscr;
    {  halt; }
      chain_return := true;             {set return flag}
      assign(workhorse,'velocity.chn'); {return control to menu program}
      chain(workhorse);

end.    {main}
```

```
(********************** HEADER AND SCREEN REVIEW PROGRAM **********************)
(*                                                                          *)
(*     This procedure is designed for looking at stored screen files and    *)
(*     data headers.  Screen files consist of Preview generated plots,      *)
(*     Transfer generated T-saft data plots and T-saft reconstructed images.*)
(*     Screen files may be Printed if desired by answering  Y  to a print   *)
(*     prompt.   Header files are files generated by Preview or Transfer.   *)
(*     When the header parameters are display they may be modified and      *)
(*     rewritten into the original files.                                   *)
(*                                                  7-1-87                  *)
(****************************************************************************)

Program   Review;

{ Global Required Definition of variables and constants }

    TYPE                                {writeheader variables}
          id = RECORD
                 nx,                    {nx= # of A-scans in one data set}
                 nz     :integer;       {nx = A-scan lenght, normally 400}
                 w,                     {w = survey width}
                 dt,                    {dt = sampling period}
                 v      :real;          {v = radar propagation velecity}
                 sp     :real;          {sp = averaged data scan-scan spacing}
                 Xavg   :integer;       {Xavg = # of A-scan averaged per data A-scan}
                 stn    :integer;       {station = data station}
                 ln     :real;          {ln = data lenght}
                 scns   :integer;       {scns =# of A-scansin the data}
                 mrk    :integer;       {delimiter between A-scans in original data}
                 zoffset:integer;        {template determined sample offset}
                 dcap   :real;           { data capacity of a disk }
                 srcfile:string[40];{scource data file}
                 spare1:real;            {template determined velocity}
                 spare2:integer;         {spare}
               END;
    VAR   { global variables }
          fname: string[40];            {name of I/O file}
          fin: string[40];              {name of input file}
          workhorse:file;               {chainfile assignee}
          fout:string[40];              {name of output file}
          which_screen:integer;         {which screen to use }
          chain_return:boolean;         {returning from chain program?}
          auto_processing:string[14];{automatic disk processing key}
          {the following characters are used for program flow control}
          char1,                        {plot intensity var in Preview}
          char2,                        {width calibtation var in Preview}
          char3,                        {print var in Preview}
          char4,                        {print var in Transfer}
          char5,                        {disk process var in Transfer}
          char6,                        {multiple processing var in T-saft}
          char7,                        {velocity stepping var in T-saft}
          char8,                        {print image var in T-saft}
          char9,                        {display format var in T-saft}
          char10,                  {spare character variables}
```

```
            char11,             {spare character variables}
            char12,             {spare character variables}
            char13,             {spare character variables}
            char14,             {spare character variables}
            char15:char;        {spare character variables}
            sparename1,
            sparename2,
            sparename3,
            sparename4:string[40];  {spare strings}
            spareI1,
            spareI2,
            spareI3,
            spare14:integer;        {spare integers}
            spareR1,
            spareR2,
            spareR3,
            spareR4:real;           {spare reals}
            info:id;                {header information}

 {the above are global variables to be passed from progran to program during
  the chain calls. These variables are to be declared exactly as above in all
  programs wether used or not.  Tailoring will be done with the spares.}


      TYPE  Num = byte;
            heappointer =  integer; {heap pointer}
      CONST nxmax = 64;             {max no of ascans per dataset}
            {aslen = 400; }           {no of data points per ascan}
      VAR   { main proc }
        scrname :string[40];
        buff: array[0..127] of num; {input parameter buffer}
        fdata: FILE;                {output disk file}
        print:char;                 {control for data plot printing}
        aslen,                      {length of radar traces, normally 400}
        n,K,I,j,l,m,                {general indices}
        marker,                     {delimiter between A-scan traces}
        navg,                       {# radar scans to average for data set}
        points,                     {number of A-scans per data set}
        station,                    {ground station reference}
        depthoff,                   {depth offset}
        scans:integer;              {number of scans}
        datcap,                     {data disk capacity}
        space,                      {A-scan spacing}
        width,                      {approx. survey width}
        fileln,lenght,              {length in bytes of radar data file}
        calspace:real;              {calculated scan spacing for saft data file}
        printscr:boolean;           {control variable for plot printing}
        input:real;                 {parameter modification input}
        vel,                        {radar velocity}
        tsamp:real;                 {sampling period}
        change:char;                {control for modifying variables}
        OK:boolean;                 {file existence flag}
        choice:char;                {choice for viewing header or screen}
        heaptop:heappointer;        {heap pointer}
```

```
{$R+}
{$V-}

{These include files are defined in the Borland Turbo Graphix Toolbox
 manual and must be included in this order.  They are used in this program
 by procedure calls coded in the program.  For further information consult
 the Borland Turbo Graphix Toolbox manual.}
  {$I c: graphix typedef.sys}                   {variable declaration}
  {$I c: graphix graphix.sys}                   {definitions and routines}
  {$I c: graphix kernel.sys}                    {support routine}
  {$I c: graphix windows.sys}                   {graphics window manipulation
                                                              routine}
(************************ OPEN FILE *****************************************)
(*                  procedure to open data file                             *)
(***************************************************************************)

procedure openfile;
  BEGIN
     assign(fdata,fname);
     {$I-} reset(fdata) {$I+};
     OK := (IOresult = 0);
  END;{procedure openfile}

(***************************** PROCEDURE SETUP *******************************)
(*                                                                           *)
(*        defines the display areas for input data drawing                   *)
(*                                                                           *)
(*****************************************************************************)
Procedure setup;
var
  st:string[6];
begin
  clrscr;
  InitGraphic;                   {initialize graphic systems}
  DefineWindow(1,0,0,79,180);    {define display window and coordinates}
  Defineworld(1,0,0,639,399);
  DefineWindow(2,0,180,79,199);  {define axis window}
  Defineworld(2,0,0,639,20);
  SelectWorld(1);                {select data window}
  SelectWindow(1);
  SetBackground(0);
  setcolorwhite;
end;     {setup}

{******************************  PRINT_SCREEN  ***************************}
(*                                                                         *)
(***************************************************************************)

procedure Print_Screen;
var
  device: FILE of integer;
  ffeed: integer;                 {code for form feed}

begin
```

```
    inline   ($55/$cd/$05/$5d);        {print screen}
    assign(device,'prn');
    ffeed := 12;
    rewrite(device);
    write(device,ffeed);               {advance to top of form}
    close(device);
  end;    {print_screen}

  (************************ PROCEDURE WRITEHEADER ******************************)
  (*                                                                          *)
  (*      write pertinent data parameters in data file header                 *)
  (*                                                                          *)
  (****************************************************************************)

  Procedure writeheader;
      begin
       reset(fdata);
       with info do                    {set header values}
            begin
                   nx := points;
                   nz := aslen;
                   w := width;
                   dt := tsamp;
                   v := vel;
                   sp := calspace;
                   Xavg := navg;
                   stn := station;
                   ln  := lenght;
                   scns := scans;
                   mrk := marker;
                   zoffset := depthoff;
                   dcap := datcap;
                   srcfile :=fin;
             END;
        blockwrite(fdata,info,(sizeof(info)+127) DIV 128);
      end;    {writeheader}

  (********************** PARAMETER INPUT **********************************)
  (*                                                                       *)
  (*                                                                       *)
  (*************************************************************************)
  Procedure parminput;
    BEGIN
       blockread(fdata,buff,1);                {read header information}
       move(buff,info,sizeof(info));
          with info do                         {set header values}
            begin
                   points := nx;
                   aslen := nz;
                   width := w;
                   tsamp := dt;
                   vel := v;
                   calspace := sp;
                   navg := Xavg;
```

```
                      station := stn;
                      lenght := ln;
                      scans := scns;
                      marker := mrk;
                      depthoff := zoffset;
                      datcap := dcap;
                      fin := srcfile;
            end;
       END; {parameter input}

      (************************ LIST HEADER VARIABLES *************************)
      (*                                                                      *)
      (************************************************************************)

      procedure list_Var;
        Begin
            writeln;
            writeln('  points   := ',points:8);
            writeln('  aslen    := ',aslen:8);
            writeln('  width    := ',width:8:3);
            writeln('  tsamp    := ',tsamp*1e9:8:4,' nS');   {note: tsamp and}
            writeln('  vel      := ',vel*1e-9:8:4,' ft/nS'); {vel are in their}
            writeln('  calspace := ',calspace:8:4);          {actual values}
            writeln('  navg     := ',navg:8);                {when in the header}
            writeln('  station  := ',station:8);
            writeln('  lenght   := ',lenght:8:0);
            writeln('  scans    := ', scans:8);
            writeln('  mark     := ',marker:8);
            writeln('  depthoff := ',depthoff:8);
            writeln('  datcap   := ',datcap:8:0);
            writeln('  fin      := ',fin);
            writeln;
         End;

      (********************** MODIFY HEADER VARIABLES *************************)
      (*                                                                      *)
      (************************************************************************)

      Procedure modify;
        Begin
            writeln;
            writeln('Enter correct values after each displayed variable,');
            writeln('press  Cr  after each entry of if values need no change.');
            writeln;
            writeln('fname := ', fname);
            write('  points   := ',points:8,'  ');         readln(points);
            write('  aslen    := ',aslen:8,'  ');          readln(aslen);
            write('  width    := ',width:8:3,'  ');        readln(width);
            input := 0;
            write('  tsamp    := ',tsamp*1e9:8:4,' nS ');  readln(input);
            if input <> 0 then tsamp := input*1e-9
                          else tsamp := tsamp;
            input := 0;
            write('  vel      := ',vel*1e-9:8:4,' ft/nS ');readln(input);
        if input <> 0 then vel := input*1e9
                      else vel := vel;
        write('  calspace := ',calspace:8:4,'  ');     readln(calspace);
        write('  navg     := ',navg:8,'  ');           readln(navg);
        write('  station  := ',station:8,'  ');        readln(station);
        write('  lenght   := ',lenght:8:0,'  ');       readln(lenght);
        write('  scans    := ', scans:8,'  ');         readln(scans);
        write('  mark     := ',marker:8,'  ');         readln(marker);
        write(' depthoff  := ',depthoff:8,'  ');       readln(depthoff);
        write('  datcap   := ',datcap:8:0,'  ');       readln(datcap);
        write('  fin      := ',fin,'  ');              readln(fin);
        if fin = '' then fin := info.srcfile;

     End;

  (************************* MAIN PROCEDURE ************************************)
  (*                                                                           *)
  (****************************************************************************)
  begin

    clrscr;
    chain_return := true;                {set flag}
    print := 'N';                        {default is no display print}
    choice := char10;                    {char10 contains header/screen selector}

   if (choice = 'H') or (choice='h')  {display header}
     then
       begin    {header}
        openfile;                        {open header file}
         parminput;                      {read the parameters}
         list_Var;                       {list all the parameters}
         write(' Modify values ? Y or [N]   ');
         read(change);
         if (change = 'y') or (change = 'Y') then
           begin
             modify;                     {modify header parameters}
             writeheader;                {rewrite header parameters into file}
          end;  {header}
     end;

    if (choice = 'S') or (choice = 's')  {display screen}
     then    {screen}
       begin
         mark(heaptop);       {save heap pointer for later memory release}
         openfile;                       {open screen files}
          setup;                         {prepare computer for graphics displav}
           loadscreen(fname);            {recall display screen}
           copyscreen;                   {make a back up copy}
           selectscreen(1);
           gotoxy(1,25);
           write(' Print display ? Y or [N] ');
           readln(print);                  {read peit option}
           if print = '' then print := 'N';
          if (print ='y') or (print = 'Y') then printscr := true
                                               else printscr := false;
          if printscr then
             begin
               swapscreen;              {redisplay original screen}
               Print_Screen;            {prints screen}
             end;
          leavegraphic;                 {deactivate gaphic mode}
          release(heaptop);             {give up heap space}
      end;   {screen}
      close(fdata);                     {close data file}

{  halt;}                               {test exit}
  assign(workhorse,'review.chn');
  chain(workhorse);                     {do back to menu}
end. { program reaview }
```

APPENDIX A

SYNTHETIC APERTURE FOCUSSING TECHNIQUE

The synthetic aperture focussing technique (SAFT) has been employed in airborne radar for many years, and has recently been applied to ultrasonic nondestructive testing (1,2,3) as well as to ground penetrating radar (4,5). Its primary application is reconstructing high resolution images of reflectors or targets.

The SAFT process functions similarly to the steering of a beam from a large array of individual antenna elements. By controlling the respective time delays applied to both pulsing the elements and to processing the received signals, the array can be steered to a desired angle for long range work. For shorter range work the array may be focussed onto a selected point in the volume to be imaged. Under this condition the transmitted energy is maximized at the focal point, and the processed composite receive signal will favor a target located at this focal point.

In the special case of SAFT, the aperture (length of the array) is synthesized by mechanically scanning a single transmit/receive antenna along the aperture and acquiring the RF returns at discrete positions. The transmitted beam is generally quite broad (a 45° cone angle is not uncommon) and transmit steering is not attempted. Rather, the broad beam angle allows target echoes to be detected from either side of the target in addition to directly above the target. The focussing technique is then applied to the detected return signals.

The most common form of SAFT imaging is implemented by shift and sum techniques and is known as time domain or T-SAFT. The stored return signals from each antenna position are shifted a precise amount in time corresponding to the particular location at which the system is being focussed. The signals are then algebraically added and the maximum amplitude is used to allocate intensity of the corresponding picture element (pixel) in the image. This process is repeated for each pixel in the planar image, thus focussing the synthetic array onto each pixel location in turn.

A common format in ground penetrating radar (GPR) imaging is to reconstruct a B-scan or elevation view in which the horizontal axis corresponds to antenna position and the vertical corresponds to depth. A test configuration showing this orientation appears in Figure 3 where the synthetic array is focussed onto the indicated pixel. For T-SAFT, the signal from each antenna position is shifted by an amount $t_i = \frac{d_o - d_i}{V}$, where;

d_o is the distance from the focal point to the surface,

d_i is the distance from the focal point to the ith antenna position, and

V is the relative velocity of propagation for radar energy in the soil.

After time shifting, the signals from all antenna positions will be in phase for a target at the focal point, and will sum reconstructively when added. This process is displayed pictorially in Figure 4. Prior to delays, the time-of-flight from individual antenna positions describes a hyperbola, while after delays, the signals are aligned in time.

As may be deduced, the reconstruction of an image with SAFT is computationally intense since the shift/sum process with 16 or 32 or more antenna signals must be repeated for each pixel in the image. A major concern of software design then becomes speed of image reconstruction. As a minimum, the time shifts for each depth can be stored in a look-up table to improve speed. Additionally, the absolute maximum amplitude of each antenna waveform sampled at the appropriate time delay may be used. In this manner summing of only one data point is required so that the complete waveforms do not need to be SAFT-processed for each pixel. Using this procedure, reconstruction times on the order of 30 seconds are possible for a 32 x 64 pixel image.

A prime feature of the imaging algorithm developed for the project is the velocity and time offset determination described in the following section. Exact knowledge of equivalent velocity is crucial to high quality of the images, and substantial image defocussing occurs for 5-10% errors in velocity.

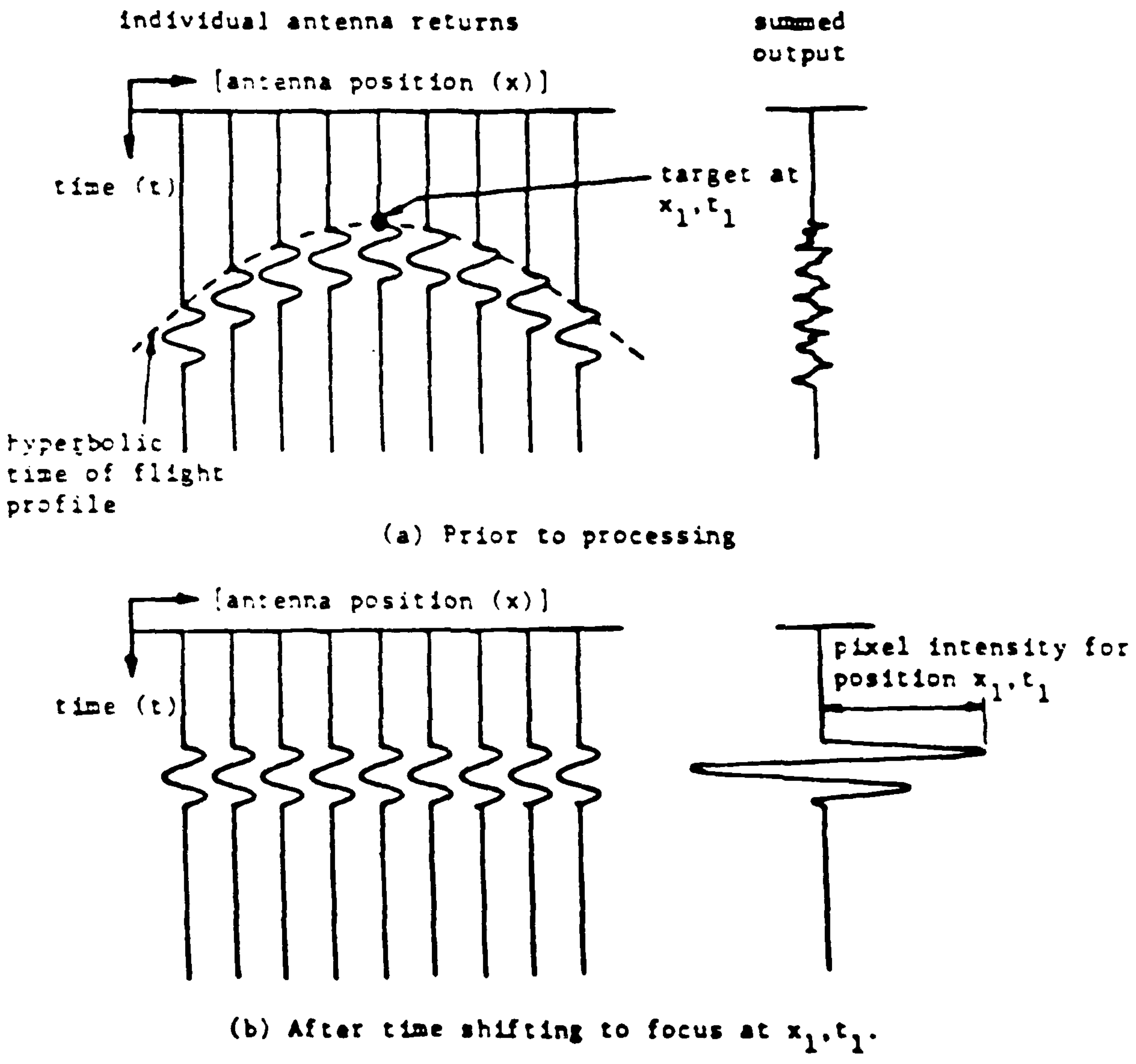

individual antenna returns
summed output
[antenna position (x)]
time (t)
target at x_1, t_1
hyperbolic time of flight profile
(a) Prior to processing
[antenna position (x)]
time (t)
pixel intensity for position x_1, t_1
(b) After time shifting to focus at x_1, t_1.

Figure A-2. Signal enhancement with SAFT delay and sum techniques.

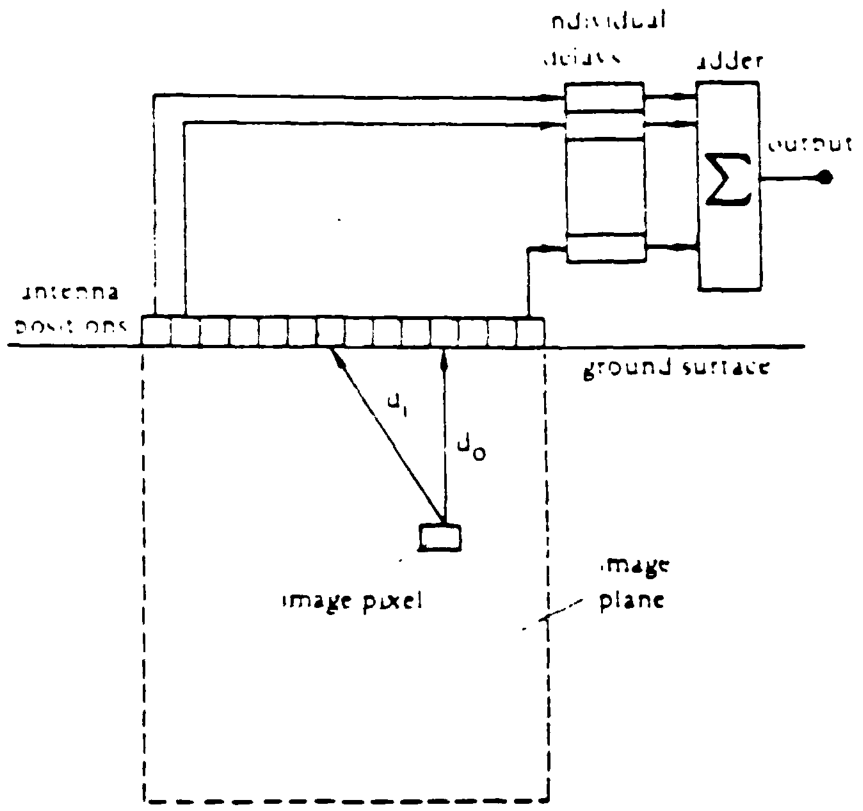


Figure A-1. Conceptual illustration of SAFT signal processing.

APPENDIX B

MINIMUM SYSTEM REQUIREMENTS

The following is a minimum system requirement to run the radar image processing software.

Hardware

IBM PC or compatible with the following:

- 8087 or 80287 mathematic coprocessor
- 512 K bytes RAM
- Two floppy disk drives
- Parallel printer port (for hard copy)
- 640 x 200 graphics capability

Dot matrix printer

- FX185 or equivalent

Software

DOS Version 3.1 including graphics

NCEL Radar Imaging Software

APPENDIX C

INPUT DATA FORMAT

A: Specification for Data Format

Technical specification of the input data:

1. Data shall be in 5 1/4 in. computer diskettes formatted in IBM PC DOS.

2. The data fill shall be written in serial eight bit bytes.

3. The serial eight bit bytes shall represent the digitized data of the radar scans stacked end to end from the beginning to the end of the survey.

4. Each radar scan shall be digitized in 400 samples or less. The first digitized point shall be zero or 255 and will be used as the start of scan marker.

5. The following criteria shall be used in the digitizing.

   a. Valid full scale 1 to 254 (0 to 255 are reserved for start of scan marker).

   b. Analog voltage 'zero' shall be equivalent to a digitized value of 127.

Operational specification of the radar data:

1. Survey should be taken with a consistent distance between scans.

2. A calibration should also be recorded such that the actual time duration represented by each digitized sample can be determined. This can be done by dividing the calibration period by the number of digitized sample points in the period.

3. Benchmarks should be made during the survey preferably five feet apart. A minimum of two markings made in each survey. This marking will be used for data width calibration during processing.

4. The data ecording should be done such that the first digitized sample corresponds to the top of ground surface.

Physical parameters such as: Traverse distance, benchmark distance, station, radar propagation velocity, etc. should be recorded in a notebook.

B: Data Header Format

The following record specifies the format of the parameter headers used in the PREVIEW output header file and the TRANSFER header. This header is 128 bytes long.

```
TYPE                              {writeheader variables}
      id = RECORD
            nx,                   {nx= # of A-scans in one data set}
            nz     :integer;      {nx = A-scan lenght, normally 400}
            w,                    {w = survey width}
            dt,                   {dt = sampling period}
            v      :real;         {v = radar propagation velecity}
            sp     :real;         {sp = averaged data scan-scan spacing}
            Xavg   :integer;      {Xavg = # of A-scan averaged per data A-scan}
            stn    :integer;      {station = data station}
            ln     :real;         {ln = data lenght}
            scns   :integer;      {scns =# of A-scansin the data}
            mrk    :integer;      {delimiter between A-scans in original data}
            zoffset:integer;      {template determined sample offset}
            dcap   :real;         { data capacity of a disk }
            srcfile:string[40];{scource data file}
            sparel:real;           (spare)
```

C: T-SAFT Data Format

The T-SAFT data format consists of the parameter header specified above followed by serial bipolar integers which represent the data from the radar survey. The offset in the unipolar byte formt of the radar data is corrected by TRANSFER to obtain the bipolar. The total data length is 57,376 bytes long.

APPENDIX D

LIST OF RUN TIME ERRORS

The following run time and input and output error messages are included here to facilitate run time fault analysis. They have been extracted from Appendix F and Appendix G of the Turbo Pascal version 3.0 manual by Borland International.

Appendix F. RUN-TIME ERROR MESSAGES

Fatal errors at run-time result in a program halt and the display of the message.

```
Run-time error NN, PC=addr
   Program aborted
```

where *NN* is the run-time error number, and *addr* is the address in the program code where the error occurred. The following contains explanations of all run-time error numbers. Notice that the numbers are hexadecimal!

01 **Floating point overflow.**

02 **Division by zero attempted.**

03 **Sqrt argument error.**
The argument passed to the Sqrt function was negative.

04 **Ln argument error.**
The argument passed to the Ln function was zero or negative.

10 **String length error.**
1) A string concatenation resulted in a string of more than 255 characters. 2) Only strings of length 1 can be converted to a character.

11 **Invalid string index.**
Index expression is not within 1..255 with *Copy*, *Delete* or *Insert* procedure calls.

90 **Index out of range.**
The index expression of an array subscript was out of range.

91 **Scalar or subrange out of range.**
The value assigned to a scalar or a subrange variable was out of range.

92 **Out of integer range.**
The real value passed to *Trunc* or *Round* was not within the *Integer* range −32768..32767.

F0 **Overlay file not found.**

FF **Heap/stack collision.**
A call was made to the standard procedure *New* or to a recursive subprogram, and there is insufficient free memory between the heap pointer (HeapPtr) and the recursion stack pointer (RecurPtr).

Appendix G
I/O ERROR MESSAGES

An error in an input or output operation at run-time results in in I/O error. If I/O checking is active (I compiler directive active), an I/O error causes the program to halt and the following error message is displayed:

```
I/O error NN, PC=addr
Program aborted
```

Where *NN* is the I/O error number, and *addr* is the address in the program code where the error occurred.

If I/O error checking is passive ({$I-}), an I/O error will not cause the program to halt. Instead, all further I/O is suspended until the result of the I/O operation has been examined with the standard function *IOresult*. If I/O is attempted before *IOresult* is called after en error, a new error occurs, possibly hanging the program.

The following contains explanations of all run-time error numbers. Notice that the numbers are hexadecimal!

01 **File does not exist.**
The file name used with *Reset*, *Erase*, *Rename*, *Execute*, or *Chain* does not specify an existing file.

02 **File not open for input.**
1) You are trying to read (with *Read* or *Readln*) from a file without a previous *Reset* or *Rewrite*. **2)** You are trying to read from a text file which was prepared with *Rewrite* (and thus is empty). **3)** You are trying to read from the logical device LST:, which is an output-only device.

03 **File not open for output.**
1) You are trying to write (with *Write* or *Writeln*) to a file without a previous *Reset* or *Rewrite*. **2)** You are trying to read from a text file which was prepared with *Reset*. **3)** You are trying to read from the logical device KBD:, which is an input-only device.

04 **File not open.**
You are trying to access (with *BlockRead* or *BlockWrite*) a file without a previous *Reset* or *Rewrite*

10 **Error in numeric format.**
The string read from a text file into a numeric variable does not conform to the proper numeric format (see page 43)

20 **Operation not allowed on a logical device.**
You are trying to *Erase*. *Rename*. *Execute*. or *Chain* a file assigned to a logical device.

21 **Not allowed in direct mode.**
Programs cannot be *Executed* or *Chained* from a program running in direct mode (i e a program activated with a **Run** command while the Memory compiler option is set).

22 **Assign to std files not allowed.**

90 **Record length mismatch.**
The record length of a file variable does not match the file you are trying to associate it with.

91 **Seek beyond end-of-file.**

99 **Unexpected end-of-file.**
1) Physical end-of-file encountered before EOF-character (Ctrl-Z) when reading from a text file **2)** An attempt was made to read beyond end-of-file on a defined file **3)** A *Read* or *BlockRead* is unable to read the next sector of a defined file Something may be wrong with the file or (in the case of *BlockRead*) you may be trying to read past physical EOF.

F0 **Disk write error.**
Disk full while attempting to expand a file. This may occur with the output operations *Write*. *WriteLn*. *BlockWrite*. and *Flush*. but also *Read* *ReadLn*. and *Close* may cause this error. as they cause the write buffer to be flushed

F1 **Directory is full.**
You are trying to *Rewrite* a file, and there is no more room in the disk directory

F2 **File size overflow.**
You are trying to *Write* a record beyond 65535 to a defined file

F3 **Too many open files.**

FF **File disappeared.**
An attempt was made to *Close* a file which was no longer present in the disk directory, e.g. because of an unexpected disk change

APPENDIX E

SUPPORT SOFTWARE

The following support software is required for making any changes to the NCEL radar data imaging software. Only DOS is needed to run the NCEL radar imaging software.

1. Turbo Pascal, Version 3.0

   By Borland International, 4585 Scotts Valley Drive, Scotts Valley, CA 95066

   This Turbo Pascal is the compiler environment on which the NCEL software has been developed. It contains the main editor and Pascal computer.

2. Turbo Graphics Tool Box, Version 1

   By Borland International (same address)

   This support software has been used to develop the high resolution graphics display.

3. Screen Sculptor

   By the Software Bottling Company of New York, New York City, NY 11378

   This software package has been used to generate the colorful menu displays and the interactive inputs of the menu parameters.

4. IBM Disk Operating System (DOS), Version 3.1

   By IBM

   The NCEL software was developed and runs on this DOS.

REFERENCES

1. S. Ganapathy, et.al., 'Analysis and Design Considerations for a Real Time System for Nondestructive Evaluation in the Nuclear Industry', Ultrasonics, November, 1982, pp. 249-256.

2. V. Schmitz, et.al., 'A New Ultrasonic Imaging System', Materials Evaluation, January, 1983, pp. 101-108.

3. S. Doctor, et.al., 'Real Time SAFT-UT for Nuclear Component Inspection', Proceedings of the 8th International Conference on NDE in the Nuclear Industry, November, 1986, available from American Society for Metals.

4. T. Davis, 'Development of a Pipe Location System', Final Report to the Gas Research Institute on Contract No. 5080-353-0335, October, 1984.

5. K. Ueno, et.al., 'An Underground Object Imaging System with Computerized Reconstruction', Proceedings of IEEE-IECEJ-AST International Conference on Acoustics, Speech and Signal Processing, April, 1986.

6. M. C. Hironaka, R. Liem, and T. J. Davis, 'Image Reconstruction with Ground Penetrating Radar Data', NO-DIG 87 International Society for Trenchless Technology, April 1987.

DISTRIBUTION LIST

ARMY CERL Library, Champaign, IL
ARMY CRREL Library, Hanover, NH
ARMY EWES Library, Vicksburg MS
DTIC Alexandria, VA
GIDEP OIC, Corona, CA
NAVFACENGCOM Code 03, Alexandria, VA
NAVFACENGCOM - CHES DIV, FPO-1PL, Washington, DC
NAVFACENGCOM - LANT DIV Library, Norfolk, VA
NAVFACENGCOM - NORTH DIV, Code 04AL, Philadelphia, PA
NAVFACENGCOM - PAC DIV, Library, Pearl Harbor, HI
NAVFACENGCOM - SOUTH DIV, Library, Charleston, SC
NAVFACENGCOM - WEST DIV, Code 04A2.2 (Lib), San Bruno, CA
PWC Code 101 (Library), Oakland, CA; Code 123-C, San Diego, CA; Code 420, Great Lakes, IL; Library (Code 134), Pearl Harbor, HI; Library, Guam, Mariana Islands; Library, Norfolk, VA; Library, Pensacola, FL; Library, Yokosuka, Japan; Tech Library, Subic Bay, RP